# S-PLUS 6.1 for Windows Supplement

May 2002

Insightful Corporation
Seattle, Washington

**Proprietary Notice**

Insightful Corporation owns both this software program and its documentation. Both the program and documentation are copyrighted with all rights reserved by Insightful Corporation.

The correct bibliographical reference for this document is as follows:

*S-PLUS 6.1 for Windows Supplement*, Insightful Corporation, Seattle, WA.

Printed in the United States.

**Copyright Notice**

Copyright © 1987-2002, Insightful Corporation. All rights reserved.

Insightful Corporation
1700 Westlake Avenue N, Suite 500
Seattle, WA 98109-3044
USA

# ACKNOWLEDGMENTS

S-PLUS would not exist without the pioneering research of the Bell Labs S team at AT&T (now Lucent Technologies): John Chambers, Richard A. Becker (now at AT&T Laboratories), Allan R. Wilks (now at AT&T Laboratories), Duncan Temple Lang, and their colleagues in the statistics research departments at Lucent: William S. Cleveland, Trevor Hastie (now at Stanford University), Linda Clark, Anne Freeny, Eric Grosse, David James, José Pinheiro, Daryl Pregibon, and Ming Shyu.

Insightful Corporation thanks the following individuals for their contributions to this and earlier releases of S-PLUS: Douglas M. Bates, Leo Breiman, Dan Carr, Steve Dubnoff, Don Edwards, Jerome Friedman, Kevin Goodman, Perry Haaland, David Hardesty, Frank Harrell, Richard Heiberger, Mia Hubert, Richard Jones, Jennifer Lasecki, W.Q. Meeker, Adrian Raftery, Brian Ripley, Peter Rousseeuw, J.D. Spurrier, Anja Struyf, Terry Therneau, Rob Tibshirani, Katrien Van Driessen, William Venables, and Judy Zeh.

# CONTENTS


**Acknowledgments** iii

**Chapter 1 Welcome** 1

- **Welcome to S-PLUS 6.1!** 2
- **Installation** 3
- **What's New in S-PLUS 6.1** 6
- **Help, Support, and Learning Resources** 8
- **Typographic Conventions** 17

**Chapter 2 CONNECT/Java** 19

- **Introduction** 20
- **Calling Java From S-PLUS** 22
- **Calling S-PLUS From Java Applications** 36

**Chapter 3 Globalization** 51

- **Introduction** 52
- **Working With Locales** 53
- **Using Extended Characters** 57
- **Importing, Exporting, and Displaying Numeric Data** 59

**Chapter 4 User-Defined Help** **63**

**Introduction** 64

**Required Software and Scripts** 66

**Creating, Editing, and Distributing a Help File** 68

**Common Text Formats** 74

**Contents of Help Files** 76

**Chapter 5 CONNECT/C++** **93**

**Introduction** 94

**Simple Examples: An Application and a Called Routine** 95

**CONNECT/C++ Class Overview** 101

**CONNECT/C++ Architectural Features** 104

**A Simple S-PLUS Interface** 115

**Index** **119**

# WELCOME 1


**Welcome to S-PLUS 6.1!** **2**

**Installation** **3**
- Supported Platforms and System Requirements 3
- Installation Instructions 3
- Running S-PLUS 4

**What's New in S-PLUS 6.1** **6**
- CONNECT/Java 6
- Globalization 6
- User-Defined Help 7

**Help, Support, and Learning Resources** **8**
- Online Help 8
- Online Manuals 11
- Tip of the Day 12
- S-PLUS on the Web 12
- Training Courses 13
- Technical Support 13
- Books Using S-PLUS 14

**Typographic Conventions** **17**

# WELCOME TO S-PLUS 6.1!

S-PLUS 6.1 is a service upgrade for S-PLUS 6, a significant new release of S-PLUS based on the latest version of the powerful, object-oriented S language developed at Lucent Technologies. S is a rich environment designed for interactive data discovery and is the only language created specifically for data visualization and exploration, statistical modeling, and programming with data.

S-PLUS 6 continues to be the premier solution for your data analysis and technical graphing needs. The Microsoft Office-compatible user interface gives you point-and-click access to data manipulation, graphing, and statistics. With S-PLUS 6 Professional, you can also program interactively using the S-PLUS programming language.

**Note**

There are two versions of S-PLUS 6 for Windows: S-PLUS 6 Professional Edition and S-PLUS 6 Standard Edition. The Standard Edition has all the features of the Professional Edition's graphical user interface but has no **Commands** or **Script** windows, no Commands History, and no access (except via a script at startup) to libraries and modules. Standard Edition users should ignore references to such features.

In a typical S-PLUS session, you can:

- Import data from virtually any source.
- View and edit your data in a convenient **Data** window.
- Create plots with the click of a button.
- Control details of your graphics to produce stunning, professional-looking output for export to your report documents.
- Perform statistical analyses from convenient dialogs in the menu system.
- Run analysis functions one at a time at the command line or in batches using the **Script** window (S-PLUS 6 Professional only).
- Create your own functions (S-PLUS 6 Professional only).
- Completely customize your user interface.

# INSTALLATION

## Supported Platforms and System Requirements

S-PLUS 6.1 for Windows is supported on the following:

- Windows 98, Windows ME, and Windows XP Professional
- Windows NT 4.0 and Windows 2000 running on Intel platforms

The minimum recommended system configuration is Pentium II/233 with 96MB of RAM. You must have at least 256MB of free disk space for the typical installation (and, if not installing on drive **C:**, an additional 75MB of free disk space on drive **C:** to unpack the distribution).

In addition, the following are required:

- Super VGA or most other Windows-compatible graphics cards and monitors with a resolution of 800x600 or better.
- One CD-ROM drive, local or networked.
- Microsoft mouse or other Windows-compatible pointing device.
- Windows-compatible printer (optional).

| **Note** |
| --- |
| S-PLUS is not supported under WIN32S (that is, Windows 3.1x), nor is it supported under Windows NT 3.51 or Windows 95. |

## Installation Instructions

To install the software:

1. Insert the CD-ROM into the CD-ROM drive.
2. If your system has CD-ROM AutoPlay enabled, the S-PLUS Setup program starts automatically. Otherwise, click the **Start** button, then **Run**, then type **x:\setup** (where **x** is the drive letter of the CD-ROM). Alternatively, double-click the **setup.exe** file in the CD-ROM drive of Windows Explorer.
3. The S-PLUS Setup window appears.

4. Follow the on-screen Setup instructions. For most installations, the default settings are recommended. It is a good idea to turn off other applications while installing S-PLUS, in particular virus checkers and screen savers.
5. During installation, you are prompted for your S-PLUS serial number. Note that the serial number format has changed dramatically in S-PLUS 6.1; you can find your new serial number on the S-PLUS 6.1 CD-ROM jewel case. The serial number you enter during installation is recorded and you can retrieve it later by selecting **Help ► About S-PLUS** from the main S-PLUS menu. If you do not enter a serial number during installation, the Trial Edition of S-PLUS 6.1 is automatically installed and time bombed to expire in 30 days.

These instructions apply to all standalone versions of S-PLUS 6.1. For complete installation instructions for the Network Edition of S-PLUS 6.1, consult the S-PLUS 6.1 release notes and the manual *S-PLUS 6 Network Installation Guide*.

Do not install this release over any existing version of S-PLUS. Instead, designate a clean installation directory for S-PLUS 6.1 and proceed with the Setup program as described above.

**Note**

If the operating system of your machine is Windows NT, Windows 2000, or Windows XP Professional, you must have full Administrator privileges to install S-PLUS. This is because the installation program must write to both the **C:\WINNT\System** folder and the registry, which is possible only if you have the correct privileges. This is a restriction imposed by InstallShield, the installation program that S-PLUS uses.

## Running S-PLUS

The S-PLUS program group contains the following options:

- **S-PLUS 6.1** starts the full GUI application.
- **S-PLUS Console** starts the command-line application only.
- **Release Notes** opens this document.
- **Fixed Bugs** opens the fixed bug list for this release.
- **Language Reference** opens the online help for the S-PLUS engine functions.

- **Using S-PLUS** opens the online help for the S-PLUS GUI.
- **Chapters Help** opens an HTML document that describes the chapter mechanism.
- **ConnectC++ Class Library Help** opens an HTML document that describes the interface available between S-PLUS and C++.
- **Migration Wizard** launches a wizard that guides you through converting projects from S-PLUS 2000 or earlier to a form compatible with S-PLUS 6.1.
- **Check for Updates** checks Insightful Corporation's Web site for patches and updates.

After installing S-PLUS, the S-PLUS program group appears as an option under **Programs** when you click the **Start** button.

To start S-PLUS with the full graphical user interface:

1. From the **Start** menu, choose **Programs**.
2. Choose the S-PLUS program group.
3. Choose **S-PLUS 6.1**.

To start S-PLUS as a console application without the GUI:

1. From the **Start** menu, choose **Programs**.
2. Choose the S-PLUS program group.
3. Choose **S-PLUS Console**.

# WHAT'S NEW IN S-PLUS 6.1

In this section, we briefly describe the new features in S-PLUS 6.1. Detailed information can be found in the relevant chapters of this supplement. For information on other enhancements and improvements in S-PLUS 6.1, see the release notes.

**Note**

This supplement includes a chapter on CONNECT/C++, a feature introduced in S-PLUS 6 as a tool for interfacing C++ with the S language. Although this chapter originally appeared in the *S-PLUS 6 for Windows Programmer's Guide*, it is being included here because it has been recently expanded to provide several examples illustrating how to use the library.

## CONNECT/Java

CONNECT/Java is now available with S-PLUS for Windows through the **winjava** library. Previously, CONNECT/Java was available only in S-PLUS for UNIX. The **winjava** library provides the `java.graph` function as well as the `.JavaMethod` functionality, which allows S-PLUS to call Java classes and Java applications to call S-PLUS.

Java is a powerful programming language with a variety of attractive features. Its strengths include a clean object-oriented design, a rich set of class libraries, and cross-platform capabilities. With CONNECT/Java, S-PLUS may be used with Java in a variety of ways. Just as S-PLUS can call C and Fortran routines, it can call Java methods to perform computations. Alternatively, a Java program can call S-PLUS to perform computations and create graphs.

## Globalization

Aspects of S-PLUS have been significantly enhanced to support globalization and locales. The concept of a *locale* is akin to **Regional Options** (or **Regional Settings**) in Microsoft Windows; setting a locale is a way of specifying which character sets, number formats, and collation sequences to use. This allows S-PLUS to support Western European character sets (ISO 8859-1) as well as 8-bit ASCII characters (ASCII codes 128-255) in character data and the names of

objects. Globalization also permits you to import, export, and display numbers written with decimal markers other than the period, and thousands separators other than the comma. S-PLUS 6.1 fully supports French and German locales.

**Note**

The implementation of locales within S-PLUS does not apply to date formats because they are handled through a separate mechanism in the `timeDate` set of functions. Also, it does not apply to messaging; all messages in S-PLUS are in the English language.

## User-Defined Help

In S-PLUS 6.1, the process for creating and distributing custom S-PLUS help files has been greatly streamlined and better documented.

# HELP, SUPPORT, AND LEARNING RESOURCES

There are a variety of ways to accelerate your progress with S-PLUS. This section describes the learning and support resources available to S-PLUS users.

## Online Help

S-PLUS offers an online HTML Help system to make learning and using S-PLUS easier. Under the **Help** menu, you will find help on using the S-PLUS graphical user interface. In addition, an extensive Language Reference provides detailed help on each function in the S-PLUS language. You can also access the Language Reference through the **Commands** window by typing `help()` at the S-PLUS prompt.

Context-sensitive help is available by clicking the **Help** buttons in dialogs or the context-sensitive **Help** button on toolbars, as well as by pressing the F1 key while S-PLUS is active.

### HTML Help

HTML Help in S-PLUS is based on Microsoft Internet Explorer and uses an HTML window to display the help files. To access HTML Help, do one of the following:

- From the main menu, choose **Help ► Available Help ► S-PLUS Help** for help on the graphical user interface.
- From the main menu, choose **Help ► Available Help ► Language Reference** for help on the S-PLUS programming language.

As shown in Figure 1.1, the HTML Help window has three main areas: the *toolbar*, the *navigation pane*, and the *topic pane*.

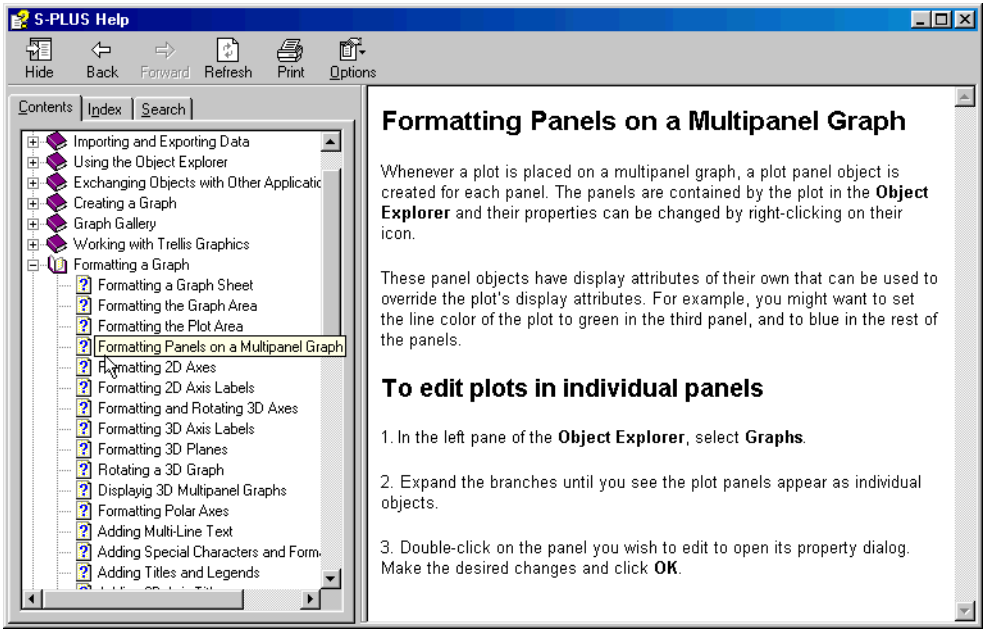

S-PLUS Help
Hide
Back
Forward
Refresh
Print
Options
Contents
Index
Search
Importing and Exporting Data
Using the Object Explorer
Exchanging Objects with Other Applicatic
Creating a Graph
Graph Gallery
Working with Trellis Graphics
Formatting a Graph
Formatting a Graph Sheet
Formatting the Graph Area
Formatting the Plot Area
Formatting Panels on a Multipanel Graph
Formatting 2D Axes
Formatting 2D Axis Labels
Formatting and Rotating 3D Axes
Formatting 3D Axis Labels
Formatting 3D Planes
Rotating a 3D Graph
Displaying 3D Multipanel Graphs
Formatting Polar Axes
Adding Multi-Line Text
Adding Special Characters and Form
Adding Titles and Legends
Formatting Panels on a Multipanel Graph
Whenever a plot is placed on a multipanel graph, a plot panel object is created for each panel. The panels are contained by the plot in the Object Explorer and their properties can be changed by right-clicking on their icon.
These panel objects have display attributes of their own that can be used to override the plot's display attributes. For example, you might want to set the line color of the plot to green in the third panel, and to blue in the rest of the panels.
To edit plots in individual panels
1. In the left pane of the Object Explorer, select Graphs.
2. Expand the branches until you see the plot panels appear as individual objects.
3. Double-click on the panel you wish to edit to open its property dialog. Make the desired changes and click OK.

**Figure 1.1:** *The S-PLUS HTML Help window.*

### Using the toolbar

Table 1.1 lists the four main buttons on the help window toolbar (in some cases, you may see more).

**Table 1.1:** *Toolbar buttons in the HTML Help window.*

| Button Name | Description |
|---|---|
| **Hide** (or **Show**) | Determines whether the navigation pane is visible. If the button is labeled **Hide**, it hides the navigation pane, expanding the topic pane to the full width of the help window. If the button is labeled **Show**, it shows the navigation pane and partitions the help window accordingly. |
| **Back** | Returns to previously viewed help topic. |

**Table 1.1:** *Toolbar buttons in the HTML Help window. (Continued)*

| Button Name | Description |
|---|---|
| **Forward** | Moves to next help topic in a previously-displayed sequence of topics. |
| **Print** | Prints the current help topic. You can also right-click in the topic pane and select **Print**, or you can right-click the topic in the **Contents** page of the navigation pane and select **Print**. |

### Using the navigation pane

The navigation pane appears on the left side of the HTML Help window. Like the help window itself, the left pane is divided into three parts: the **Contents** page, the **Index** page, and the **Search** page:

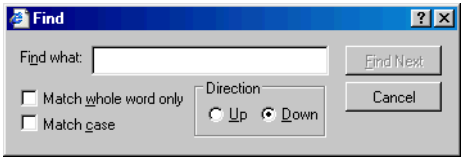
- The **Contents** page organizes help topics by category so related help files can be found easily. These categories appear as small book icons, labeled with the name of the category. To open a category, double-click the icon or label. To select a topic within the category, double-click its question-mark icon or the topic title.
- The **Index** page lists available help topics by keyword. Keywords are typically function names for S-PLUS language functions and topic names for graphical user interface topics. Type a word in the text box and HTML Help dynamically finds the keyword that most closely matches it. Click **Display** (or double-click the selected title) to display the help topic.
- The **Search** tab provides a full-text search for the entire help system. Type the word or phrase you want to find in the text box and click the **List Topics** button or press ENTER. HTML Help displays in the list box all help files containing that keyword. Select the desired topic and click **Display** (or double-click the selected title) to display the help topic.

### Using the topic pane

The topic pane appears on the right side of the help window and displays the help topics you choose. It usually appears with both vertical and horizontal scroll bars, but you can expand the HTML Help window to increase the width of the right pane. Many help files are too long to be fully displayed in a single screen, so choose a convenient height for your HTML Help window and then use the vertical scroll bars to scroll through the text.

The right pane contains a search-in-topic feature. To use it:

1. Type CTRL-F to open the **Find** dialog (this dialog is a feature of HTML Help inherited from Internet Explorer).




2. Type your search string in the text field labeled **Find what**.
3. Click **Find Next**.

## Help at the Command Line

When working from the S-PLUS command line, you can obtain help for any S-PLUS function using the `help` or `?` functions. For example, to open the help file for `anova`, simply type:

```
> help(anova)
```

or

```
> ?anova
```

## Help in the Script Window

To obtain help for a command when working in a **Script** window, simply highlight the command and press F1.

# Online Manuals

In addition to this supplement, the booklet *Getting Started with S-PLUS 6 for Windows*, the *User's Guide*, the *Programmer's Guide*, and both volumes of the *Guide to Statistics* are available online. *Getting Started with S-PLUS 6* provides a tutorial introduction to the product and so is particularly useful for those new to S-PLUS.

To view a manual online, choose **Help ▶ Online Manuals** from the main menu and select the desired title.

| Note |
| --- |
| The online manuals are viewed using Adobe Acrobat Reader. It is generally useful to turn on *bookmarks* under the **View** entry of the menu bar while using Acrobat Reader, rather than rely on the contents at the start of the manuals. Bookmarks are always visible and can be expanded and collapsed to show just chapter titles or to include section headings. |

## Tip of the Day

To help speed your progress in S-PLUS, a handy Tip of the Day appears by default each time you start the program; see Figure 1.2.

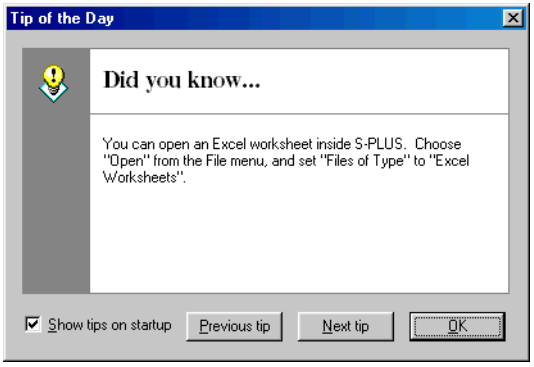

Tip of the Day
Did you know...
You can open an Excel worksheet inside S-PLUS. Choose "Open" from the File menu, and set "Files of Type" to "Excel Worksheets".
Show tips on startup
Previous tip
Next tip
OK

**Figure 1.2:** *A Tip of the Day.*

You can access the S-PLUS Tips of the Day at any time by choosing **Help ▶ Tip of the Day** from the main menu. If you prefer to turn off this feature, simply clear the **Show tips on startup** check box in the dialog.

## S-PLUS on the Web

You can find S-PLUS on the World Wide Web at **http://www.insightful.com/support**. In these pages, you will find a variety of information, including:

- Product information.
- FAQ pages.
- The most recent service packs.

- Training course information.
- Information on classroom use and related educational materials.

## Training Courses

Insightful Corporation's Educational Services offers a number of courses designed to quickly make you efficient and effective at analyzing data with S-PLUS. The courses are taught by professional statisticians and leaders in statistical fields. Courses feature a hands-on approach to learning, dividing class time between lecture and online exercises. All participants receive the educational materials used in the course, including lecture notes, supplementary materials, and exercise data. For further information, see

**http://www.insightful.com/services/training.asp**.

## Technical Support

### North, Central, and South America

Contact Technical Support at Insightful Corporation:

**Telephone**: 206.283.8802 or 1.800.569.0123, ext. 235, Monday-Friday, 6:00 a.m. PST (9:00 a.m. EST) to 5:00 p.m. PST (8:00 p.m. EST)

**Fax**: 206.283.8691

**E-mail**: support@insightful.com

**Web**: **http://www.insightful.com/support**

### All Other Locations

Contact the European Headquarters of Insightful Corporation:

Christoph Merian-Ring 11, 4153 Reinach, Switzerland

**Telephone**: +41 61 717 9340

**Fax**: +41 61 717 9341

**E-mail**: info@ch.insightful.com

## Books Using S-PLUS

### Introductory

Burns, P. (1998). *S Poetry*. Download for free from

**http://www.seanet.com/~pburns/Spoetry**

Krause, A. and Olson, M. (2000). *The Basics of S and S-PLUS* (2nd ed.). Springer-Verlag, New York.

Lam, L. (2001). *An Introduction to S-PLUS for Windows*. CANdiensten, Amsterdam.

Spector, P. (1994). *An Introduction to S and S-PLUS*. Duxbury Press, Belmont, CA.

### Statistics and Data Analysis (General)

Alzola, C.F. and Harrell, F.E. (2001). *An Introduction to S-PLUS and the Hmisc and Design Libraries*. Download for free from

**http://hesweb1.med.virginia.edu/biostat/s/doc/splus.pdf**

Atkinson, A., and Riani, M. (2000). *Robust Diagnostic Regression Analysis*. Springer-Verlag.

Bowman, A. and Azzalini, A. (1997). *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-PLUS Illustrations*. Clarendon Press.

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S*. Chapman and Hall.

Crawley, M. (2002). *Statistical Computing: An Introduction to Data Analysis Using S-PLUS*. Wiley.

Davison, A.C. and Hinkley, D.V. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press.

Efron, B. and Tibshirani, R.J. (1994). *An Introduction to the Bootstrap*. Chapman & Hall.

Everitt, B. (1994). *A Handbook of Statistical Analyses Using S-PLUS*. Chapman & Hall, London.

Härdle, W. (1991). *Smoothing Techniques with Implementation in S*. Springer-Verlag, New York.

Hastie, T.J. and Tibshirani, R.J. (1990). *Generalized Additive Models*. Chapman & Hall.

Hastie, T.J., Tibshirani, R.J., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer-Verlag, New York.

Huet, S., et al. (1996). *Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS Examples.* Springer-Verlag, New York.

Marazzi, A. (1992). *Algorithms, Routines and S Functions for Robust Statistics.* Wadsworth & Brooks/Cole, Pacific Grove, CA.

Newton, H.J. and Harvill, J.L. (1997). *StatConcepts: A Visual Tour of Statistical Ideas.* Duxbury Press.

Nolan, D. and Speed, T.P. (2000). *Stat Labs: Mathematical Statistics Through Applications.* Springer-Verlag.

Pinheiro, J.C. and Bates, D.M. (2000). *Mixed-Effects Models in S and S-PLUS.* Springer-Verlag, New York.

Schumaker, R.E. and Akers, A. (2001). *Understanding Statistical Concepts Using S-PLUS.* Lawrence Erlbaum Associates, Mahwah, NJ.

Venables, W.N. and Ripley, B.D. (1999). *Modern Applied Statistics with S-PLUS* (3rd ed.). Springer-Verlag, New York.

Wilcox, R. (1997). *Introduction to Robust Estimation and Hypothesis Testing.* Academic Press.

### Statistics and Data Analysis (Specialized)

Bruce, A. and Gao, H.-Y. (1996). *Applied Wavelet Analysis with S-PLUS.* Springer-Verlag, New York.

Everitt, B. and Rabe-Hesketh, S. (2001). *Analyzing Medical Data Using S-PLUS (Statistics for Biology and Health).* Springer-Verlag.

Harrell, F.E. (2001). *Regression Modeling Strategies with Applications to Linear Models, Logistic Regression, and Survival Analysis.* Springer-Verlag.

Kaluzny, S.P., Vega, S.C., Cardoso, T.P., and Shelly, A.A. (1997). *S+SPATIALSTATS User's Manual.* Springer-Verlag, New York.

Meeker, W. and Escobar, L. (1998). *Statistical Methods for Reliability Data.* John Wiley & Sons, Inc.

Millard, S.P. and Krause, A. (2001). *Applied Statistics in the Pharmaceutical Industry.* Springer-Verlag, New York.

Millard, S.P. and Neerchal, N.K. (2000). *Environmental Statistics with S-PLUS.* CRC Press.

Selvin, S. (1998). *Modern Applied Biostatistical Methods: Using S-PLUS*. Oxford University Press.

Therneau, T.M. and Grambsch, P.M. (2000). *Modeling Survival Data*. Springer-Verlag, New York.

### Graphical Techniques

Chambers, J.M., Cleveland, W.S., Kleiner, B., and Tukey, P.A. (1983). *Graphical Methods for Data Analysis*. Duxbury Press, Belmont, CA.

Cleveland, W.S. (1993). *Visualizing Data*. Hobart Press, Summit, NJ.

Cleveland, W.S. (1994). *The Elements of Graphing Data*. Hobart Press, Summit, NJ.

### S-PLUS Programming

Becker, R.A., Chambers, J.M., and Wilks, A.R. (1988). *The New S Language*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Chambers, J.M. (1998). *Programming with Data*. Springer-Verlag, New York.

Venables, W.N. and Ripley, B.D. (2000). *S Programming*. Springer-Verlag, New York.

# TYPOGRAPHIC CONVENTIONS

Throughout this supplement, the following typographic conventions are used:

- `This font` is used for S-PLUS expressions and code samples.
- **This font** is used for elements of the S-PLUS user interface, for operating system files and commands, and for user input in dialog fields.
- *This font* is used for emphasis and book titles.
- CAP/SMALLCAP letters are used for key names. For example, the Shift key appears as SHIFT.
- When more than one key must be pressed simultaneously, the two key names appear with a hyphen (-) between them. For example, the key combination of SHIFT and F1 appears as SHIFT-F1.
- Menu selections are shown in an abbreviated form using the arrow symbol (►) to indicate a selection within a menu, as in **File ► New**.

# CONNECT/JAVA

# 2


**Introduction** **20**
- Java Tools 20
- Conventions 20
- Release Notes 20
- Example Files 20

**Calling Java From S-PLUS** **22**
- Static Fields 22
- Static Methods 25
- Class Files 26
- Instance Methods 27
- Managing Java Object Instances 29

**Calling S-PLUS From Java Applications** **36**
- Running the Java Program 36
- Evaluating S-PLUS Commands 37
- Using `SplusDataResult` Objects 37
- Example Applications 41

# INTRODUCTION

CONNECT/Java is a tool for interfacing Java with S-PLUS. Java is a powerful programming language with a variety of attractive features. Its strengths include a clean object-oriented design, a rich set of class libraries, and cross-platform capabilities.

S-PLUS may be used with Java in a variety of ways. Just as S-PLUS can call C and Fortran routines, it can call Java methods to perform computations. Alternatively, a Java program can call S-PLUS to perform computations and create graphs.

S-PLUS Analytic Server adds the ability to call S-PLUS remotely from a Java application. This enables your Java program to reside on a client machine and call S-PLUS on a remote server to perform computations and create graphs.

## Java Tools

The **winjava** library installation includes a copy of the Java 2 runtime environment (JRE). This runtime environment includes the Java virtual machine and related classes necessary to run any Java 2-compliant compiled Java code. Users with compiled Java code need no additional tools to use their code with S-PLUS.

Users writing new code need a Java 2 development environment such as the Java 2 JDK. For S-PLUS 6.0, the desired JDK is version 1.3. This is freely available for Windows from Sun. See **www.javasoft.com** for details. Licensing restrictions prevent Insightful from redistributing the Java 2 JDK with S-PLUS.

## Conventions

Within this chapter, **SHOME** refers to the directory containing the S-PLUS installation. In the current implementation of the **winjava** library, the library must be installed within **SHOME**. See the **readme.txt** file for details.

## Release Notes

See the **readme.txt** file in the **winjava** directory for installation instructions, details on basic usage, and a list of known problems.

## Example Files

The Java code samples for the examples discussed here are included in ***.java** files in the **SHOME\library\winjava\examples** directory. The **java** directory distributed as part of the **winjava** library contains

the file **examples.jar**. This file includes the byte-compiled ***.class** files for all of the ***.java** files. As these classes are available to Java by default, it is not necessary to compile or install the Java code to run the examples.

# CALLING JAVA FROM S-PLUS

S-PLUS can exchange information with Java via static fields and static methods. Currently, S-PLUS does not have a way to directly call nonstatic methods, which would require the S-PLUS engine to keep track of references to particular instances of Java objects.

Functionality that is available through instance methods (nonstatic methods) may be accessed by creating static Java methods that take care of the details of creating Java instances, tracking them, and calling methods on them. These static methods may then be called from S-PLUS.

## Static Fields

Java objects may be passed from Java to S-PLUS through static fields. The S-PLUS function `.JavaField` returns a field value given the following information:

- The name of the Java class containing the field. The package name may be forward slash or period delimited.
- The name of the field.
- The Java Native Interface (JNI) field descriptor for the field.
- The optional `client` argument.

For example, the following call returns the value of PI:

```
> .JavaField("java/lang/Math", "PI", "D")
```

The `.JavaField` function is the primary mechanism for transferring values from Java into S-PLUS.

### Field Descriptors

The Java Native Interface (JNI) is the standard interface for passing information between Java and native C code. In JNI, the type of a field is declared using a *field descriptor*. For example, an `int` field is represented with `I`, a `float` field with `F`, a `double` field with `D`, a `boolean` field with `Z`, and so on.

The descriptor for a reference type such as `java.lang.String` begins with the letter `L`, is followed by the *class descriptor*, and is terminated by a semicolon. (The class descriptor is typically the class name delimited with slashes.) The field descriptor for a `String` is `Ljava/lang/String;`.

Descriptors for array types consist of the `[` character, followed by the descriptor of the component type of the array. For example, `[I` is the descriptor for the `int[]` field type.

S-PLUS uses the field descriptors to communicate field types to Java. S-PLUS can access values of type `void`, `int`, `short`, `long`, `boolean`, `float`, `double`, `byte`, `char`, and `String`. It can also access arrays of these types. Table 2.1 presents each supported Java type, the JNI field descriptor, and the corresponding S-PLUS type. All S-PLUS atomic types except `complex` have Java equivalents.

**Table 2.1:** *JNI field descriptors.*

| JNI Field Descriptor | Java Type | S-PLUS Type |
|---|---|---|
| `V` | `void` | `NULL` |
| `I` | `int` | `integer` |
| `S` | `short` | `integer` |
| `J` | `long` | `integer` |
| `Z` | `boolean` | `logical` |
| `F` | `float` | `single` |
| `D` | `double` | `double (numeric)` |
| `B` | `byte` | `raw` |
| `C` | `char` | `character` |
| `Ljava/lang/String;` | `String` | `character` |

The latest version of S-PLUS extends the connection to allow access of any Java object. If the object is an array of objects or an object whose class implements `java.util.Collection` (such as `Vector` or `ArrayList`), then the result is an S-PLUS list. For any other type of object, the `toString()` method is used to return a string.

The `javap` utility included with the Java 2 SDK prints out the JNI field descriptors for a class. For example:

```
% javap -s -p java.lang.Math
Compiled from Math.java
public final class java.lang.Math extends java.lang.Object
{
    public static final double E;
        /*   D   */
    public static final double PI;
        /*   D   */
    private static java.util.Random randomNumberGenerator;
        /*   Ljava/util/Random;   */
    private static long negativeZeroFloatBits;
        /*   J   */
    private static long negativeZeroDoubleBits;
        /*   J   */
...
```

**Integer Conversions**

S-PLUS integers are equivalent to C longs. They are either 32-bit or 64-bit in size, depending upon whether the operating system is 32-bit or 64-bit. In Java, `shorts` are 16-bit, `ints` are 32-bit, and `longs` are 64-bit. All three types are converted to S-PLUS integers.

**Special Values**

S-PLUS has the special values `NA`, `Inf`, and `-Inf` to represent missing values, positive infinity, and negative infinity, respectively. Java has the special values `NaN`, `POSITIVE_INFINITY`, and `NEGATIVE_INFINITY` for these cases. These special values are mapped appropriately when values are transferred.

**`client` Argument**

The `client` argument is used by S-PLUS Analytic Server. Its value is irrelevant in the **winjava** library. The argument is specified in some of the examples so that they are correct for S-PLUS Analytic Server. Otherwise, it may be omitted.

**Field Examples**

The `java.lang.Math` class contains a few interesting static fields that we can access from S-PLUS:

```
> .JavaField("java/lang/Math", "PI", "D")
[1] 3.141593
> .JavaField("java/lang/Math", "E", "D")
[1] 2.718282
```

```
> .JavaField("java/lang/Double", "NaN", "D")
[1] NA
```

## Static Methods

Scalars and arrays of primitives may be passed from S-PLUS to a static Java method, and a Java object may be returned from the Java method to S-PLUS. The S-PLUS function `.JavaMethod` takes the following arguments:

- The name of the Java class. The package name may be forward slash or period delimited.
- The name of the method.
- The JNI method descriptor indicating the types of the arguments.
- Optionally, one or more values used as the arguments to the method.
- Optional `client` argument, which is used similarly to the argument in `.JavaField()`.

For example, the following call returns 2 to the power of 10:

```
> .JavaMethod("java/lang/Math", "pow", "(DD)D", 2, 10)
[1]  1024
```

Note that `.JavaMethod` automatically converts the values to doubles based on the signature of the method. Unlike the S-PLUS functions `.C` and `.Fortran`, the programmer does not need to assure that the values are of a particular type via calls to `as.double()`.

### Method Descriptors

The JNI method descriptors are formed from the field descriptors for each method argument and the field descriptor for the return value. The argument types appear first and are surrounded by one pair of parentheses. Argument types are listed in the order in which they appear in the method declaration. If a method takes no arguments, this is represented by empty parentheses. The method's return type is placed immediately after the right closing parenthesis.

For example, `(I)V` denotes a method that takes an `int` argument and has return type `void`, which is returned in S-PLUS as a `NULL`.

A method that takes a `String` argument and returns a `String` is denoted by `(Ljava/lang/String;)Ljava/lang/String;`.

Arrays are again indicated with `[` character, followed by the descriptor of the array element type. A method that takes a `String` array and returns `void` has the method descriptor `([Ljava/lang/String;)V`.

The `javap` utility included with the Java 2 SDK prints out the JNI method descriptors for a class. For example:

```
% javap -s -p java.lang.Math
Compiled from Math.java
public final class java.lang.Math extends java.lang.Object
{
 ...
    public static double toDegrees(double);
        /*   (D)D   */
    public static double toRadians(double);
        /*   (D)D   */
}
```

### `client` Argument

The `client` argument is used by S-PLUS Analytic Server. Its value is irrelevant in the **winjava** library.

### Simple Examples

Here are some examples using `.JavaMethod`:

```
> .JavaMethod("java/lang/Math", "round", "(D)J", pi)
[1] 3
> .JavaMethod("java/lang/Math", "round", "(F)I", pi)
[1] 3
> .JavaMethod("java/lang/Math", "random", "()D")
[1] 0.6300195
> .JavaMethod("java/lang/System", "getProperty",
+ "(Ljava/lang/String;)Ljava/lang/String;", "os.name")
[1] "Windows2000"
```

The `StaticMethodsExample` class provides various examples of how to retrieve values from Java. The Java code for this class is in **SHOME\library\winjava\examples**.

## Class Files

In order for the Java virtual machine to find Java classes, the class files must be in one of the locations where Java looks for files. The location may be specified using either the Java classpath or the Java extensions mechanism.

The main difference between these two approaches appears to be a difference in strictness of security such that any extension class can be instantiated via reflection, while objects in the classpath may not always be instantiated with reflection. This is unlikely to be an issue for most users. However, if you get a "Class Not Found" exception when using the classpath mechanism and reflection, you may want to try the extension mechanism instead.

### Java Classpath

Java finds class files that are in a location specified in the Java classpath. If the class files are in a "jar" file, the location is the full path to the jar file, including the name of the file. If the class files are in a directory, the location is the full path to the directory.

The classpath may be modified from the Windows command prompt by setting the `CLASSPATH` environment variable:

```
set CLASSPATH="D:\My Dir"
```

The classpath may also be specified using the `-classpath` argument to `java`.

### Java Extensions

Java automatically finds any jar files placed in **java\jre\lib\ext**. This is the default location for Java extensions.

Other directories may be added to the list of extension directories with the `-Djava.ext.dirs` argument to `java`. If this is specified, it must be a path that also lists the standard **java\jre\lib\ext** directory.

## Instance Methods

Instance methods are methods that reference a particular Java object rather than static class information. Instance methods may be called by writing static methods that manage creation and tracking of the necessary instances.

### Random Numbers Example

The class `java.util.Random` provides objects that represent pseudorandom number streams. We can obtain random numbers from Java by creating a class that instantiates a `java.util.Random` object and provides methods for getting values from the stream.

```
import java.util.Random;
public class RandomWrapperExample {
  static Random ran = new Random();
```

```
    public static double nextDouble(){
        return ran.nextDouble();
    }

    public static int nextInt(){
        return ran.nextInt();
    }
}
```

After compiling this class (typically by invoking `javac` on the Java source file) and adding its location to the classpath, we can call these methods from S-PLUS:

```
> .JavaMethod("RandomWrapperExample", "nextDouble", "()D")
[1] 0.9606592
> .JavaMethod("RandomWrapperExample", "nextInt", "()I")
[1] 4026360078
```

### File Chooser Example

The `JFileChooser` class provides a file chooser dialog that is useful for locating and selecting a file. We can create a Java class that launches this dialog and returns the path to the selected file. The S-PLUS function for invoking the file chooser lets the user specify the starting location for the browser.

If the dialog does not appear when this function is run, focus has probably changed in such a way that the dialog is behind other applications. Minimize the other applications to locate the dialog.

The S-PLUS function is:

```
fileChooserExample <- function(startPath = getenv("PWD")){
  .JavaMethod("FileChooserExample", "showFileChooser",
      "(Ljava/lang/String;)Ljava/lang/String;",
      startPath)
}
```

The Java class definition is:

```
import javax.swing.JFileChooser;

public class FileChooserExample {
  public static String showFileChooser(String startPath){
    JFileChooser fileChooser = new JFileChooser(startPath);
    fileChooser.setDialogTitle("Select File");
```

```
    int exitStatus = fileChooser.showDialog(null, "OK");

    String selectedFileName = "";
    if (exitStatus == JFileChooser.APPROVE_OPTION)
      selectedFileName =
        fileChooser.getSelectedFile().getAbsolutePath();

    return selectedFileName;
  }
}
```

## Managing Java Object Instances

S-PLUS does not directly attempt to track instances of Java objects. However, instances may be created using Java code, and identifiers to these instances may be passed back to S-PLUS for use in tracking the objects for further manipulation.

The basic technique is to use a static `Vector`, `HashTable`, or other type of collection in the Java class. When a new instance is created, it is placed in the collection and a key into the table is passed back to S-PLUS. This key may be passed into a static method that then finds the object in the table and applies the relevant method to the object.

In this example, we create a Java class representing a person and family relationship information about the person. When we create an object for the person, we specify the person's first and last names. We then have methods to indicate family relationships between individuals. In particular, we can indicate an individual's mother and father using `setMother()` and `setFather()` methods. These methods modify the individual to note the parenting information and also modify the parent's object to note that the individual is their child. We can retrieve information about the individual using the `getInfo()` method.

Before showing the Java code, let's see how these methods would be used from within S-PLUS. In this example, we use `.JavaMethod()` directly at the S-PLUS prompt. We could create S-PLUS functions to call these routines in order to avoid having to specify the class name and method signature each time we want to use a method.

First, we create three `FamilyMember` objects representing a mother, father, and son. The `newFamilyMember()` method creates an object in Java and returns an integer ID corresponding to the object.

```
> momId <- .JavaMethod("FamilyMember", "newFamilyMember",
+ "(Ljava/lang/String;Ljava/lang/String;)I",
+ "Sue", "Jones")
> dadId <- .JavaMethod("FamilyMember", "newFamilyMember",
+ "(Ljava/lang/String;Ljava/lang/String;)I",
+ "Tom", "Jones")
> sonId <- .JavaMethod("FamilyMember", "newFamilyMember",
+ "(Ljava/lang/String;Ljava/lang/String;)I",
+ "Skip", "Jones")
```

Next, we use the `setMother()` and `setFather()` methods to establish the relationship between the parents and the son. Note that we are using the identifiers returned above.

```
> .JavaMethod("FamilyMember", "setMother", "(II)Z",
+ sonId, momId)
[1] T
> .JavaMethod("FamilyMember", "setFather", "(II)Z",
+ sonId, dadId)
[1] T
```

Now that we have created the objects and specified their relationship, we can use `getInfo()` to examine the objects. The `getInfo()` method uses the family relationship information to determine the names of parents and children for the individual.

```
> .JavaMethod("FamilyMember", "getInfo",
+ "(I)[Ljava/lang/String;", sonId)
[1] "Name: Skip Jones"  "Mother: Sue Jones"
[3] "Father: Tom Jones"
> .JavaMethod("FamilyMember", "getInfo",
+ "(I)[Ljava/lang/String;", dadId)
[1] "Name: Tom Jones"  "Mother: Unknown"
[3] "Father: Unknown"  "Child: Skip Jones"
```

The Java code for the `FamilyMember` class is straightforward. We present it here with comments describing the key points.

```
import java.util.Vector;

public class FamilyMember {
  /* This class is an example of creating and modifying a
     dynamic collection of instances using static methods.
  */

  // Track instances by keeping a static Vector of
  // instances.  We will add each FamilyMember object to
  // this Vector when it is created, and return its index
  // in this Vector as the key for accessing the object.

  static Vector members = new Vector();

  // Instance variables.  We get each person's first and
  // last name when the object is created.  Methods are then
  // used to specify their mother and father.  When the
  // person is specified as a mother or father, we know
  // they have a child, which we also track.

  String firstName, lastName;
  FamilyMember mother, father;
  Vector children = new Vector();

  /* Constructor */

  public FamilyMember (String first, String last){
    firstName = first;
    lastName = last;
  }

  /* Instance methods */
  /* Java routines would call these */

  // public methods to get names

  public String getFirstName(){
    return firstName;
  }
```

```
public String getLastName(){
  return lastName;
}

// public methods to set and get parents

public void setMother(FamilyMember mom){
  mother = mom;
  mother.addChild(this);
}

public void setFather(FamilyMember dad){
  father = dad;
  father.addChild(this);
}

public FamilyMember getMother(){
  return mother;
}

public FamilyMember getFather(){
  return father;
}

// private method to add child when parent set

private void addChild(FamilyMember kid){
  children.add(kid);
}

// public method to get children

public Vector getChildren(){
  return children;
}

/* Static methods */
/* S-PLUS would call these */
```

```
// static method to create a family member and return
// an ID to track them

public static int newFamilyMember(String first,
    String last){
  FamilyMember newMember = new FamilyMember(first, last);

  // Add new instance to list of members
  members.add(newMember);

  // Use the position in the members vector as an ID
  return (members.size() -1);
}

// private method to check that ID in legal range

private static boolean checkId(int id){
  boolean status = true;

  if (id < 0 || id > (members.size()-1)){
    // Could throw exception, but we'll just print a
    // message
    System.out.println("Error:  ID out of range");
    status = false;
  }
  return status;
}

// static methods to specify mother and father based on ID

// The basic steps in these methods are:
//   1) Check that the ID is within the range of ID's.
//   2) Get the object from the members Vector.
//   3) Cast the object to a FamilyMember object.
//   4) Apply the relevant non-static method to the object
//
// If the ID is out of range we return false.  Otherwise
// we return true.
```

```
public static boolean setMother(int personId, int momId){
  boolean status = true;
  if (checkId(personId) && checkId(momId)){
    FamilyMember person =
      (FamilyMember) members.get(personId);
    FamilyMember mom = (FamilyMember) members.get(momId);
    person.setMother(mom);
  }
  else
    status = false;
  return status;
}

public static boolean setFather(int personId, int dadId){
  boolean status = true;
  if (checkId(personId) && checkId(dadId)){
    FamilyMember person =
      (FamilyMember) members.get(personId);
    FamilyMember dad = (FamilyMember) members.get(dadId);
    person.setFather(dad);
  }
  else
    status = false;
  return status;
}

// static method to get information about a family member

public static String [] getInfo(int id){
  if (!checkId(id))
    return new String [] {"Name: Unknown",
      "Mother: Unknown", "Father: Unknown"};

  FamilyMember person = (FamilyMember) members.get(id);
  FamilyMember mom = person.getMother();
  FamilyMember dad = person.getFather();
  Vector kids = person.getChildren();

  String [] info = new String [3 + kids.size()];
```

```
    info[0] = "Name: " + person.getFirstName() + " " +
      person.getLastName();

    if (mom==null)
      info[1] = "Mother: Unknown";
    else
      info[1] = "Mother: " + mom.getFirstName() + " " +
        mom.getLastName();

    if (dad==null)
      info[2] = "Father: Unknown";
    else
      info[2] = "Father: " + dad.getFirstName() + " " +
        dad.getLastName();

    for (int i = 0; i < kids.size(); i++){
      FamilyMember aKid = (FamilyMember) kids.get(i);
      if (!(aKid==null)){
        info[3+i] = "Child: " + aKid.getFirstName() + " " +
          aKid.getLastName();
      }
    }

    return info;
  }
}
```

# CALLING S-PLUS FROM JAVA APPLICATIONS

S-PLUS may be called from Java to perform computations and create graphics. In this section, we describe the primary Java classes for communicating between Java and S-PLUS. The discussion of classes is followed by a variety of examples.

The `SplusUserApp` class provides a simple way for a Java application to connect to S-PLUS. It contains static methods that Java applications can call to generate results and graphics. S-PLUS graphs created with the `java.graph` graphics device can be embedded within Java application windows.

## Running the Java Program

To run a Java program in Microsoft Windows, specify the full path to the **java** executable and the name of the Java class. This may be done at a Windows command prompt or in a Windows shortcut.

To call S-PLUS, the location of **SHOME** must also be specified using the Java property `splus.shome`. For example:

```
"D:\Program Files\Insightful\Splus61\java\jre\bin\java"
   -Dsplus.shome="D:\Program Files\Insightful\Splus61"
   TextOutputExample
```

In order for the S-PLUS engine DLL to load properly, the directory containing this DLL and related DLLs must be in the Windows PATH. This directory is the **SHOME\cmd** directory, with a value such as:

```
D:\Program Files\Insightful\Splus61\cmd
```

The PATH can be set using the **Environment Variables** dialog. To open this dialog, choose **Start ▶ Control Panel ▶ System**. In the **System Properties** dialog, click the **Advanced** tab, then click the **Environment Variables** button.

## Evaluating S-PLUS Commands

S-PLUS expressions can be evaluated with the following `SplusUserApp` methods:

```
public static SplusDataResult eval(String cmd);
public static SplusDataResult eval(String cmd,
  boolean Output, boolean Result, boolean Errors,
  boolean Warnings, boolean Expr);
```

The `cmd` parameter contains an expression to be evaluated by S-PLUS. The resulting `SplusDataResult` object contains the result of evaluating this expression.

The expression should be a syntactically complete S-PLUS expression. An `SplusIncompleteExpressionException` is thrown if the expression is not complete.

The additional parameters `Output`, `Result`, `Errors`, `Warnings`, and `Expr` allow the user to specify what elements to include in the `SplusDataResult`. These may be specified as `false` to avoid the overhead of passing unnecessary elements of the output over the data channel. The default is to include `Result`, `Errors`, and `Warnings`.

## Connection Threads

The first time `SplusUserApp.eval()` is called, it starts the S-PLUS engine process and several Java threads. It takes a few seconds for the process to complete.

These Java threads remain active until the S-PLUS engine is shut down, so it is typically necessary to call `System.exit(0)` to exit the user Java application. Even if the application's main thread is finished, Java will wait forever for the S-PLUS connection threads to finish.

## Using `SplusDataResult` Objects

An object deriving from `SplusDataResult` is produced by the `eval()` method of `SplusUserApp` and contains the result of evaluating an S-PLUS expression. This type of object is also produced by a call to `evalDataQuery()` when using the `SplusSession` interface, described later in this chapter.

The results of a query are formalized to have the following fields:

- `expression`: A string representing the query that was processed.

- `output`: A string representing the output that the query produced. This is the output that would be printed if the command were evaluated as standard input. This output may be printed to standard out or to a Java text area.
- `error`: A string representing the error output that the query produced.
- `warning`: An array of strings representing warnings that the query produced.

If the query produces a data result, it has an additional field:

- `result`: An array of values of some primitive type determined by the query. This is an array of a standard atomic type, for example, `boolean`, `byte`, `double`, `float`, `long`, or `String`. This is the mechanism for passing results of computations back to Java.

The `SplusDataResult` parent class contains the first four fields but lacks the `result` field. The following classes inherit from the `SplusDataResult` class and contain a `result` field that is an array of the appropriate type:

```
SplusBooleanDataResult
SplusByteDataResult
SplusDoubleDataResult
SplusFloatDataResult
SplusLongDataResult
SplusStringDataResult
```

These are the only types presently supported in accessing the S-PLUS analytic engine from Java. The correspondence between Java types and S-PLUS types is shown in Table 2.2.

**Table 2.2:** *Java and S-PLUS correspondence.*

| Java | S-PLUS |
|---|---|
| `boolean` | `logical` |
| `byte` | `raw` |
| `double` | `numeric` |

**Table 2.2:** *Java and S-PLUS correspondence. (Continued)*

| Java | S-PLUS |
|---|---|
| `float` | `single` |
| `long` | `integer` |
| `string` | `character` |

The programmer must ensure that queries produce only a single vector of one of these primitive types. The analytic engine then automatically constructs a data result object with a class that matches the query result type.

If the query fails to produce a result, an `SplusDataResult` object is returned that does not contain a `result` field. If the query returns a more complex object than a vector, a warning message is printed to the engine's standard output stream and an `SplusDataResult` object is returned that does not contain a `result` field.

The fields in data result objects are private and are available only via accessor methods. All data result objects include the following methods:

```
public String getExpression();
public String getOutput();
public String getError();
public String [] getWarning();

public Object getData();
public boolean [] getBooleanData() throws
    SplusBadDataException;
public byte [] getByteData() throws
    SplusBadDataException;
public double [] getDoubleData() throws
    SplusBadDataException;
public float [] getFloatData() throws
    SplusBadDataException;
public long [] getLongData() throws
    SplusBadDataException;
public String [] getStringData() throws
    SplusBadDataException;
```

Generally, only one of the `getxxxData` methods returns correct data, depending on the type of the data result that is available. If the type of the data result available does not match the type of data requested in the `get` operation, an `SplusBadDataException` is thrown with a string describing the condition.

For instance, suppose you are expecting a vector of `longs` as the result of a query. Then in a simplified situation, you would use the code:

```
SplusDataResult result = evalDataQuery("/* some query */");
long [] values = result.getLongData();
```

Suppose further that by some mistake your query produced a vector of `doubles` instead of a vector of `longs`. Then, at runtime, the call to `result.getLongData` would generate an `SplusBadDataException` with the string:

```
"long" data was requested, but existing data is "double"
```

Under some circumstances, you may not care initially about the type of return data. For example, you may be constructing a collection of result vectors that you want to store as a hash-table of Java objects. The actual type of the data would be handled at a later time, using Java's `instanceof` operator.

For such situations, the `SplusDataResult` class (and its derivatives) have the method

```
public Object getData();
```

which returns the result vector as a generic Java object. The `getData()` method returns `NULL` if the object has no data and an array of primitives otherwise. This is useful when you intend to put the result in a Java collection object such as a `Vector` rather than immediately using the result as a primitive array of a known type.

In addition to the methods discussed above, the following boolean methods are available to determine whether the fields of the `SplusDataResult` object have non-null entries:

```
public boolean hasExpression();
public boolean hasOutput();
public boolean hasError();
public boolean hasWarning();
```

The `SplusDataResult` class and the classes that inherit from it are intended to form a complete set of result classes reflecting the capabilities currently offered in interacting with the S-PLUS analytic engine.

## Graph Components

The S-PLUS `java.graph()` device displays S-PLUS graphics within a Java component. By calling `SplusUserApp` methods, a Java application can embed a `java.graph()` display component within a Java window. S-PLUS graphs are then displayed in this component by evaluating S-PLUS graphics commands.

S-PLUS typically allows multiple graphics devices to be open at once, and multiple graphics devices can be opened while evaluating a single expression. When running the S-PLUS graphical user interface, these appear in different windows. Currently, `SplusUserApp` only supports a single graph window embedded in a specific `JComponent`.

The following `SplusUserApp` method returns the `JComponent` containing the graph window:

```
public static JComponent getGraph();
```

This component contains a multi-page graph display that can be embedded within another window. The component corresponds to the last `java.graph()` device that was opened. When a new `java.graph()` device is opened, this graph is cleared and no longer has any association with the previous device.

## Example Applications

In each of the following examples, we create a simple Java class that communicates with S-PLUS using `SplusUserApp`. Java code for these examples is in **SHOME\library\winjava\examples**.

The example classes that call S-PLUS from Java are:

```
CorrelationExample
DataFrameArrayListExample
GraphButtonsExample
ImageCalculatorExample
RandomNormalExample
ResultSetExample
TextOutputExample
```

The **java** directory distributed as part of the **winjava** library contains the file **examples.jar**. This file includes the ***.class** files for all of the ***.java** files.

These examples may be run with no additional configuration by entering the following type of expression at a Windows command prompt:

```
"D:\Program Files\Insightful\Splus6\java\jre\bin\java"
  -Dsplus.shome="D:\Program Files\Insightful\Splus6"
  TextOutputExample
```

## Text Output

The `TextOutputExample` application generates the numbers 1 to 100 and returns them formatted for printing.

```
import com.insightful.splus.*;

public class TextOutputExample {
  public static void main(String [] args){
    try {
      String expression = "1:100";
      System.out.println("Sending expression " +
        expression);

      // Get just the text output back
      SplusDataResult result =
        SplusUserApp.eval(expression + "\n",
          true, false, false, false, false);

      System.out.println("Result from S-PLUS:");
      System.out.println(result.getOutput());
    }
    catch (Exception e){
      System.out.println(e);
    }

    System.exit(0);
  }
}
```

## Returning Values

The `RandomNormalExample` application generates 100 random normal values and passes them back to Java. It then prints the number of values, first value, and last value.

```
import com.insightful.splus.*;

public class RandomNormalExample {
  public static void main(String [] args){
    try {
      String expression = "rnorm(100)";

      // Get just the result output back
      SplusDataResult result =
        SplusUserApp.eval(expression + "\n",
          false, true, false, false, false);

      // Get the double values.  We know the S-PLUS function
      // rnorm() returns doubles.  If it did not, the
      // try/catch mechanism would catch the
      // SplusBadDataException.

      double [] randomValues = result.getDoubleData();

      System.out.println("Generated " +
        randomValues.length +
        " random normal values.");
      System.out.println("First value: " + randomValues[0]);
      System.out.println("Last value: " +
        randomValues[randomValues.length - 1]);
    }
    catch (Exception e){
      System.out.println(e);
    }

    System.exit(0);
  }
}
```

## Passing Values to S-PLUS

The `SplusDataResult` class provides a way to get values from S-PLUS. We can pass values needed for a computation from Java to S-PLUS in two ways.

The first way is to include the values in the string passed to `SplusUserApp.eval()`. This is the simplest approach when we are specifying a few parameters to a function, as in the previous examples.

The second way is to store the values in static fields and have the S-PLUS function query the values of these fields. The `.JavaField()` function accesses the field value directly. Alternatively, we might use a `.JavaMethod()` call to a static method that extracts the field value. The `CorrelationExample` application uses the `.JavaField()` approach.

```
import com.insightful.splus.*;

public class CorrelationExample {

  // static fields to pass values to S-PLUS
  public static double [] xValue;
  public static double [] yValue;

  // Java function calling S-PLUS to compute correlation
  // of two double arrays.  This can be called from other
  // classes.
  //
  // Throw an IllegalArgumentException if x or y
  // inappropriate.
  // Pass on other exceptions if they are thrown.

  public static double getCorrelation(double [] x,
    double [] y) throws IllegalArgumentException,
      SplusBadDataException, java.rmi.RemoteException {

   if (x == null || x.length == 0 ||
     y == null || y.length == 0)
     throw (new IllegalArgumentException(
         "Argument is null or length zero."));
```

```
    xValue = x;
    yValue = y;

    // Define the S-PLUS expression to use.
    // Note we use \" for quotes within the expression.

    String expression = "cor(" +
      ".JavaField(\"CorrelationExample\", " +
      "\"xValue\", \"[D\"), client = T)," +
      ".JavaField(\"CorrelationExample\", " +
      "\"yValue\", \"[D\", client = T))";

    SplusDataResult result =
      SplusUserApp.eval(expression + "\n");

    return result.getDoubleData()[0];
  }

  public static void main(String [] args){
    // Create some double arrays and get their correlations.

    double [] a = {2.1, 3.5, 8.4, 5.3, 6.8};
    double [] b = {9.8, 3.7, 2.1, 8.3, 6.5};

    System.out.println(
      "Getting correlation of two double arrays.");

    try {
      double d = CorrelationExample.getCorrelation(a, b);
      System.out.println("Correlation is: " + d);
    }
    catch (Exception e){
      System.out.println(e);
    }
    System.exit(0);
  }

}
```

## Embedding Graphs in a Custom GUI

The `GraphButtonsExample` application launches a window containing a graph region and buttons that generate graphs when pressed.

```
import com.insightful.splus.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class GraphButtonsExample {

  static ActionListener buttonActionListener;

  public static void main(String [] args){

    // Get the graph component
    JComponent splusGraph = SplusUserApp.getGraph();

    // Create the window
    JFrame window = new JFrame("Graph Buttons Example");

    // Create buttons which query S-PLUS to produce graphs

    buttonActionListener = new ActionListener(){
      public void actionPerformed(ActionEvent e){
         String cmd = e.getActionCommand() + "\n";
         try {
             SplusUserApp.eval(cmd, false, false,
                  false, false, false);
         }
         catch (Exception ex) {
             System.out.println(ex);
         }
      }
    };

    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new BoxLayout(buttonPanel,
      BoxLayout.Y_AXIS));
    buttonPanel.add(makeButton("plot(sin(1:10))"));
    buttonPanel.add(makeButton("plot(rnorm(100))"));
```

```
    buttonPanel.add(makeButton(
     "print(example.normal.qq())"));
    buttonPanel.add(makeButton("show colors, linetypes",
       "{image(matrix(data=1:100,nrow=10,ncol=10));" +
       "for (a in 1:50) { ang <- a*0.06; " +
       "lines(c(5,5+10*cos(ang))," +
       "c(5,5+10*sin(ang)), col=a) };for (a in 1:50) { " +
       "ang <- -a*0.06; lines(c(5,5+10*cos(ang)), " +
       "c(5,5+10*sin(ang)), lty=a) }}"));
    buttonPanel.add(makeButton(
      "for (x in 1:10) plot(1:x)"));

    // Create an Exit button

    JButton exitButton = new JButton("Exit");
    exitButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae){
        System.exit(0);
      }
    });

    // Also exit if the window is closed by the user

    window.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){
        System.exit(0);
      }
    });

    // Add the elements to the content pane
    Container contentPane = window.getContentPane();
    contentPane.setLayout(new BorderLayout());
    contentPane.add(splusGraph, BorderLayout.CENTER);
    contentPane.add(buttonPanel, BorderLayout.EAST);
    contentPane.add(exitButton, BorderLayout.SOUTH);

    // Show the window
    window.setSize(600, 400);
    window.show();
  }
```

```
  // Utility methods to create a button
  private static JButton makeButton(String splusCode){
    return makeButton(splusCode, splusCode);
  }

  private static JButton makeButton(String buttonName,
    String splusCode){
    JButton b = new JButton(buttonName);
    b.setActionCommand(splusCode);
    b.addActionListener(buttonActionListener);
    return b;
  }
}
```

### Sending a JDBC ResultSet to S-PLUS

Java provides the JDBC standard for accessing data from a database. In JDBC, a `ResultSet` represents data returned from doing a particular query to a particular database. The `ResultSetUtilities` class in the `com.insightful.splus.util` package provides support for creating an S-PLUS `data.frame` from a `ResultSet`.

This class is used by registering a `ResultSet` under a particular name. The S-PLUS function `javaGetResultSet("name")` can then create a `data.frame` containing the values in the `ResultSet`.

The `ResultSetExample` example shows how to register and access a `ResultSet`. In this example, we use a `LocalResultSet` object. This is an inner class defined to make the example self-contained and is not presented here.

```
import com.insightful.splus.*;
import com.insightful.splus.util.ResultSetUtilities;
import java.sql.*;
import java.io.*;
import java.math.BigDecimal;

public class ResultSetExample {
  public ResultSetExample() {

      LocalResultSet theSet = new LocalResultSet();

      // Register the ResultSet with ResultSetUtilities.
      ResultSetUtilities.register("mySetId", theSet);
```

```
        // Save ResultSet data in S-PLUS as "test.df".

        // Specify client = T so ResultSet will be found
        // in this JVM when in client/server mode.

        System.out.println(
          "Transferring the ResultSet to test.df in S-PLUS.");

        String expr = "assign(\"test.df\", " +
           "javaGetResultSet(\"mySetId\", client = T), " +
           "where = 1)\n";

        SplusDataResult result = SplusUserApp.eval(expr);

        if (result.hasError())
           System.out.println(result.getError());

        if (result.hasWarning()) {
           String [] warnings = result.getWarning();
           for (int i = 0; i < warnings.length; i++)
              System.out.println(warnings[i]);
        }

        // Query S-PLUS to print test.df to show it's there
        System.out.println("Printing test.df in S-PLUS.");

        expr = "if (exists(\"test.df\")) print(test.df) else
           print(\"test.df does not exist\")\n";

        result = SplusUserApp.eval(expr, true, false, false,
            false, false);
        System.out.println(result.getOutput());
    }

    public static void main(String [] args) {
        new ResultSetExample();
        System.exit(0);
    }
}
```

### Sending Multiple Columns to Java

The `SplusDataResult` mechanism provides access to a single vector of values. Multiple calls to `eval()` can be used to retrieve multiple columns of data from a `data.frame`.

Java has no direct equivalent to a `data.frame`. However, a collection such as a `Vector` or `ArrayList` is available for storing multiple arrays of data. These arrays can then be used in other Java routines, such as writing to a database via JDBC.

The `DataFrameArrayListExample` shows how to create an `ArrayList` containing the columns of a `data.frame`.

### Transferring a Graph to the Client

All of the file operations performed by S-PLUS are done on the server file system. Often it is desirable to have a file saved on the client file system. The `FileUtilities` class in `com.insightful.splus.util` provides a `transferBytes()` function that is useful for passing a file over file streams. Typically, one file stream is created in the client JVM, and the other is obtained from the `SplusSession` using `getFileInputStream()` or `getFileOutputStream()`.

### A Simple Application

The examples above are focused at performing particular tasks. The `ImageCalculatorExample` combines various techniques presented above to create an application that lets the user type in S-PLUS expressions that are executed to create text and graphics output.

# GLOBALIZATION

# 3


**Introduction** **52**

**Working With Locales** **53**
- Setting a Locale 53
- Changing a Locale 55
- Changing the Collation Sequence 56

**Using Extended Characters** **57**
- In Variable Names 57
- In PostScript Output 58

**Importing, Exporting, and Displaying Numeric Data** **59**
- Importing and Exporting Data 59
- Displaying Data 62

# INTRODUCTION

New in S-PLUS 6.1 are a number of enhancements designed to improve global installation and use:

- New functions have been added that allow you to specify a locale to be used by S-PLUS. Specifically, S-PLUS 6.1 now supports French and German locales.
- Sorting and collating functions have been modified to support different locales.
- The use of 8-bit characters (Latin1/Western European character set) has been enabled. The 8-bit characters can now be used in variable names and will properly display in data.
- Functions that import, export, and display numeric data have been modified to support different locales.
- S-PLUS Setup has been improved to test for local settings and install localized DLLs as necessary.

In the sections that follow, we describe these enhancements in greater detail.

# WORKING WITH LOCALES

The concept of a "locale" in S-PLUS is akin to that of **Regional Options** (or **Regional Settings**) in Windows. A locale determines which characters are considered printable or alphabetic, the format for numerals, and the collating sequence of characters.

| **Note** |
| --- |
| The implementation of locales in S-PLUS does not cover date formats (date formats are handled through a separate mechanism) and messaging (all messages in S-PLUS are in the English language). |

The default locale is `C`, the locale used by the C language (7-bit ASCII characters and U.S.-style numbers). The `C` locale conforms to the behavior of earlier versions of S-PLUS, that is:

- No accented characters are displayed (backslash octal instead).
- The decimal marker is a period.
- The sort order is all uppercase followed by all lowercase.

In European-based locales, however, S-PLUS prints accented characters and uses a more natural sort order ("`a`", then "`A`", then "`à`" (`"\340"`), then "`À`" (`"\300"`)). In continental-based locales, the decimal marker is a comma. In English-based (and French Canadian) locales, the decimal marker is a period.

Not every locale is supported on every computer. To see which locales are supported on a computer running a version of Windows, look in **Control Panel ► Regional and Language Options**.

## Setting a Locale

Setting a locale gives you a way to tell S-PLUS which character sets, number formats, and collation sequences to use. The primary function for setting a locale is the `Sys.setlocale()` function. This function allows you to set `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC`, or `LC_TIME`.

### In the Commands Window

To use the default regional settings for locale, open the **Commands** window and type the following:

```
> Sys.setlocale(locale="")
```

This command tells S-PLUS to follow your Windows **Regional Options** (or **Regional Settings**).

| **Hint** |
| --- |
| If you put the desired `Sys.setlocale()` command into the **\local\S.init** file under your S-PLUS home directory, the command will be run automatically each time you launch S-PLUS. |

### In the General Settings Dialog

On the **Startup** page of the **Options ▶ General Settings** dialog, the fields for controlling regional settings have been replaced with a single check box called **Set Region-specific defaults**, as shown in Figure 3.1. (This option is not selected by default.)

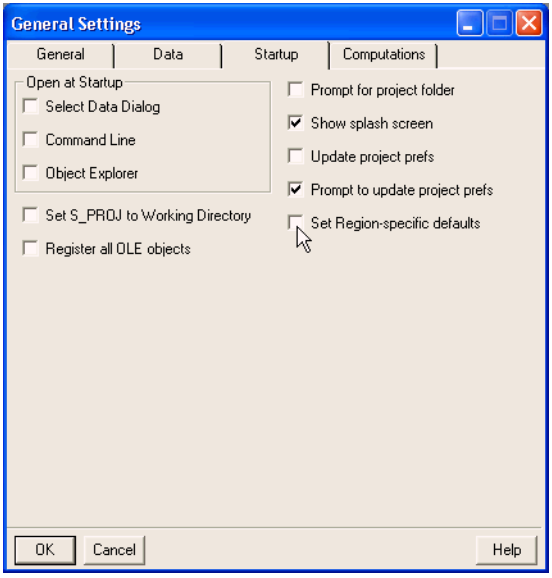



**Figure 3.1:** *The **Startup** page of the **General Settings** dialog.*

Selecting this check box sets S-PLUS's default locale according to your Windows **Regional Options** (or **Regional Settings**) in effect at the time of startup.

- If this check box is selected, the next time you start S-PLUS it sets the `time.in.format`, `time.out.format`, and `time.zone` options (shown in **.Options**) according to your Windows **Regional Options** (or **Regional Settings**). It also gets the currently selected Windows regional language setting and calls `Sys.setlocale(locale=`*`language`*`)` where *`language`* is the setting from Windows.

**Note**

When you select this option, you must restart S-PLUS for the change to take effect.

- If this check box is *not* selected, the `time.zone` option is set to the default of GMT, and `Sys.setlocale()` is set to the `C` locale.

**Note**

Selecting the **Set Region-specific defaults** check box also includes setting `LC_NUMERIC`, which causes numbers to print with a comma as the decimal marker in regions where that is the convention. This may break existing code that relies on the format of printed numbers.

If you select the **Set Region-specific defaults** option but want to use the period as the decimal marker instead, add the command `Sys.setlocale("LC_NUMERIC","C")` to the **\local\S.init** file under your S-PLUS home directory.

## Changing a Locale

Use the `Sys.setlocale()` and `Sys.getlocale()` functions to set and get locales, respectively. Use the `Sys.withlocale()` function if you want to evaluate an expression within a specified locale.

For example:

```
> Sys.getlocale()                    # Default
[1] "C"
> Sys.setlocale(locale="en_US")
[1] "en_US"
```

```
> Sys.getlocale()
[1] "en_US"
```

## Changing the Collation Sequence

"Collation sequence" refers to the ordering of characters by the `sort()` and `order()` functions. For example, in the `C` locale, uppercase (capital) letters collate before lowercase letters. In almost all other locales, however, collation sequence ignores case.

For example:

```
> Sys.setlocale(locale="C")
[1] "C"
> sort(c("as", "Axe"))
[1] "Axe" "as"
> Sys.setlocale(locale="English (United States)")
[1] "English_United States.1252"
> sort(c("as", "Axe"))
[1] "as"  "Axe"
```

Changing `LC_COLLATE` affects the way S-PLUS interprets the ordering of character data and thus can be used to change the behavior of functions like `sort()` and `order()`.

# USING EXTENDED CHARACTERS

S-PLUS 6.1 accepts more "alphabetic" characters than in earlier versions. This set of "alphabetic" characters is fixed and is the set of characters considered alphabetic in ISO 8859-1 (Latin1/Western European character set).

The set of characters that are displayed as octal codes changes depending on the current locale (specifically, `LC_CTYPE`). For example, in all locales that use Latin characters, `"\341"` is displayed as "á". This affects how functions such as `format()`, `print()`, and `cat()` display character data.

## In Variable Names

Because S-PLUS now supports Western European character sets (ISO 8859-1), you can include 8-bit ASCII characters (ASCII codes 128-255) within character data and within the names of S-PLUS objects. For example, you can include names such as Furtwängler and García Íñiguez in your character data and they will properly display. S-PLUS uses your locale settings to determine the appropriate character set to use in printing output.

The characters allowed in S-PLUS object names are:

```
alphanumerics <- c(".", "0", "1", "2", "3", "4", "5", "6",
  "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H",
  "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S",
  "T", "U", "V", "W", "X", "Y", "Z", "a", "b", "c", "d",
  "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
  "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
  "\212", "\214", "\216", "\232", "\234", "\236", "\237",
  "\300", "\301", "\302", "\303", "\304", "\305", "\306",
  "\307", "\310", "\311", "\312", "\313", "\314", "\315",
  "\316", "\317", "\320", "\321", "\322", "\323", "\324",
  "\325", "\326", "\330", "\331", "\332", "\333", "\334",
  "\335", "\336", "\337", "\340", "\341", "\342", "\343",
  "\344", "\345", "\346", "\347", "\350", "\351", "\352",
  "\353", "\354", "\355", "\356", "\357", "\360", "\361",
  "\362", "\363", "\364", "\365", "\366", "\370", "\371",
  "\372", "\373", "\374", "\375", "\376", "\377")
```

The characters not allowed in names are:

```
non.alphanumerics <- c("\001", "\002", "\003", "\004",
  "\005", "\006", "\007", "\b", "\t", "\n", "\013", "\014",
  "\r", "\016", "\017", "\020", "\021", "\022", "\023",
  "\024", "\025", "\026", "\027", "\030", "\031", "\032",
  "\033", "\034", "\035", "\036", "\037", " ", "!", "\"",
  "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-",
  "/", ":", ";", "<", "=", ">", "?", "@", "[", "\\", "]",
  "^", "_", "`", "{", "|", "}", "~", "\177", "\200",
  "\201", "\202", "\203", "\204", "\205", "\206", "\207",
  "\210", "\211", "\213", "\215", "\217", "\220", "\221",
  "\222", "\223", "\224", "\225", "\226", "\227", "\230",
  "\231", "\233", "\235", "\240", "\241", "\242", "\243",
  "\244", "\245", "\246", "\247", "\250", "\251", "\252",
  "\253", "\254", "\255", "\256", "\257", "\260", "\261",
  "\262", "\263", "\264", "\265", "\266", "\267", "\270",
  "\271", "\272", "\273", "\274", "\275", "\276", "\277",
  "\327", "\367")
```

**Note**

The set of characters allowed or disallowed in object names is not affected by the current locale. The functions `deparse()`, `parse()`, `dump()`, `dput()`, `source()`, `data.dump()`, and `data.restore()` are also unaffected by the choice of locale.

## In PostScript Output

The `postscript()` function now uses the Latin1 encoding of its standard fonts by default. This means you can more easily use non-English, Western European characters. If you previously used octal escapes like `"\267"` to get characters in the upper half of the standard PostScript encoding (`"\267"` was the bullet character), you must either change such code (`"\200"` is now the bullet) or use the arguments `setfont=ps.setfont.std` and `bullet=ps.bullet.std` in calls to `postscript()`. The Symbol and Zapf family of fonts are not changed. The Latin1 encoding is not quite the ISO 8859-1 standard, in that the bullet character was added at position `"\200"`.

# IMPORTING, EXPORTING, AND DISPLAYING NUMERIC DATA

S-PLUS 6.1 supports importing, exporting, and displaying numeric data written in regional notation. This means you can import, export, and display numbers written using decimal markers and thousands separators other than the period and comma, respectively. For example, you can specify a comma as your decimal marker and a period as your digit-grouping symbol, or you can use a period as the decimal marker and an apostrophe as your digit-grouping symbol.

This feature is supported for the functions `scan()`, `print()`, `read.table()`, and `write.table()`. It is also supported for tick labels in graphics and for display in the **Data** window. Note, however, that it is *not* supported for use within S-PLUS expressions, within `xlab` and `ylab` in graphics, or by default within the `importData()` and `exportData()` functions.

**Hint**

The default values for `xlab` and `ylab` do not use the numeric locale information because the default values are made with `deparse()`. However, if you make up your labels with `as.character()` or `paste()`, the locale information *is* used.

## Importing and Exporting Data

When importing and exporting data, you can either use the command-line functions or the import/export dialogs.

### In the Commands Window

The functions `importData()` and `exportData()` now have two additional arguments for use in reading and writing ASCII numbers:

- The `decimal.point` argument controls the single character used to mark the decimal place in ASCII numbers. The default is the period (`.`).
- The `thousands.separator` argument controls the single character used as the digit-grouping symbol in ASCII numbers. The default is the comma (`,`).

You can also specify which locale to use within `importData()` and `exportData()` by setting the argument `use.locale` to `T` (the default is `use.locale=F`.)

### In the Import/ Export Dialogs

Both the **Import From File** dialog and the **Export To File** dialog provide two new fields for specifying the decimal character and the digit-grouping symbol. (See Figure 3.2 and Figure 3.3.)

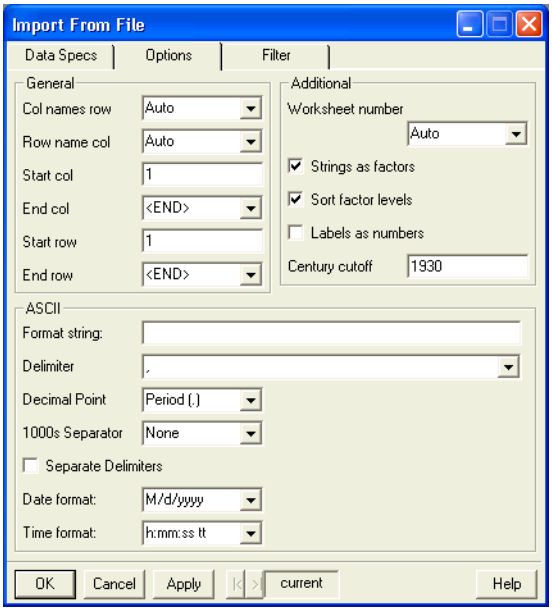



**Figure 3.2:** *The **Options** page of the **Import From File** dialog.*

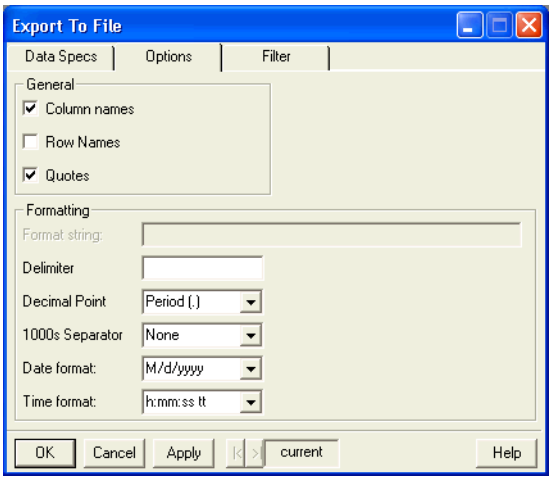



**Figure 3.3:** *The **Options** page of the **Export To File** dialog.*

On the **Options** page, make your selections as follows:

- **Decimal Point** Select **Comma (,)** or **Period (.)** from the drop-down list. The default is **Comma (,)**.
- **1000s Separator** Select **Comma (,)**, **None**, or **Period (.)** from the drop-down list. The default is **None**.

| **Note** |
| --- |
| These two fields are grayed out if the selected file format is not an ASCII file type. Note also that neither option is affected by the selection of delimiter or by any locale or regional settings. |

Two additional changes have been made to the import/export dialogs:

- A **Semicolon (;)** selection has been added to the **Delimiter** field drop-down list.
- On the **Data Specs** page, the list of ASCII file formats has been updated to the following:
  - **ASCII file - whitespace delim (asc; dat; txt; prn)**
  - **ASCII file - whitespace delim (comma-decimal) (asc; dat; txt; prn)**
  - **ASCII file - comma delimited (csv)**

- **ASCII file - semicolon delimited (comma-decimal) (csv)**
- **ASCII file - user-defined delimiter (asc; dat; txt; prn)**

**Note**

**ASCII file - delimited**, **ASCII file - space delimited**, and **ASCII file - tab delimited** have been removed from the list. However, the function `guiImportData()` will still accept these strings as valid values for its `FileTypes` argument.

The **ASCII file - semicolon delimited (comma-decimal) (csv)** option sets **Delimiter** to **Semicolon (;)**, selects the **Separate Delimiters** check box, and sets **Decimal Point** to **Comma (,)**.

The **ASCII file - whitespace delim (comma-decimal) (asc; dat; txt; prn)** option behaves like the **ASCII file - whitespace delim (asc; dat; txt; prn)** option but also sets **Decimal Point** to **Comma (,)**.

## Displaying Data

The decimal marker used when displaying or reading numbers changes depending on `LC_NUMERIC`. For example, in many European locales, the number pi is written 3,14159 (approximately). This affects how functions such as `format()`, `print()`, `cat()`, `write.table()`, and `html.table()` display numeric data. It also affects how `as.numeric()` and `as.character()` convert between strings and numbers. Other functions like `cut()` and `factor()` (in the labeling of levels) are also affected.

# USER-DEFINED HELP

# 4


**Introduction** **64**

**Required Software and Scripts** **66**
- Downoading and Running Cygwin 66
- Downloading and Running HTML Help Workshop 67
- The **build_chm_tools** Directory 67

**Creating, Editing, and Distributing a Help File** **68**
- Step 1: Creating the Help File 68
- Step 2: Editing the Help File 69
- Step 3: Executing the Script 70
- Step 4: Checking the **.chm** File 73
- Step 5: Distributing the **.chm** File 73
- Errors 73

**Common Text Formats** **74**

**Contents of Help Files** **76**
- Descriptions of Fields 76
- Special Help Files 90

# INTRODUCTION

When an S-PLUS function is made available system-wide, a help file should always accompany it. Without a help file, use of the function is likely to be limited to those in close contact with the person who wrote it.

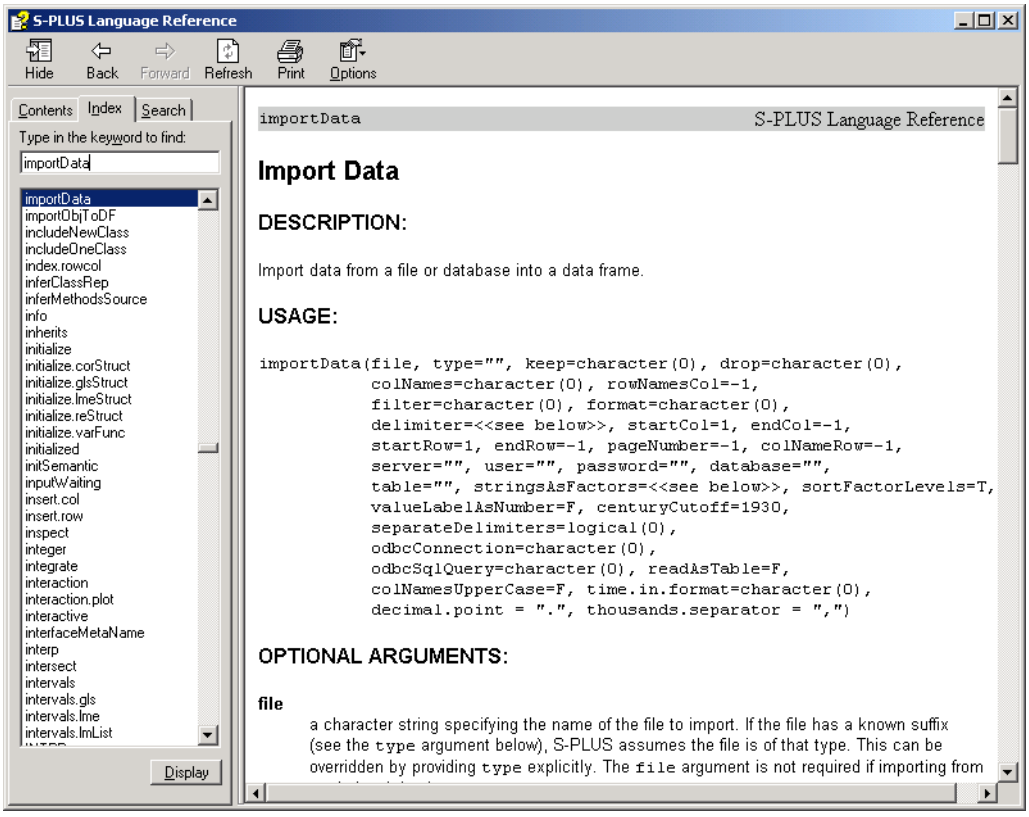



**Figure 4.1:** *An example of a compiled HTML Help project,* ***splus.chm****.*

Starting with S-PLUS 6, the S Language version 4 provides the flexibility of creating your own SGML (Standard Generalized Markup Language) help files to document your user-defined functions in S-PLUS for Windows. Using the built-in functions available in your S-PLUS installation plus two other downloads, you can distribute your user-created help files by creating a **.chm**, a compiled help file. This **.chm** file is created by compiling a HTML

Help project, and HTML Help is the online help standard from Microsoft used in S-PLUS. Figure 4.1 shows an example using **splus.chm**, which displays help files for functions in S-PLUS.

The S-PLUS function `prompt` generates a help file outline for any S-PLUS function or data object you specify. The outline includes the correct syntax for the function, as well as the current definition of the function, and headings under which you can supply the following information:

- What the function does.
- A brief description of the function.
- A description of each argument, with the option of specifying both required and optional arguments.
- A description of the value returned by the function.
- *Side effects*, which are any effects of the function besides the return of a value.
- The *method* behind the function; that is, how the function works.
- Any references to the literature.
- Cross-references to other help files.
- Keywords. Keywords are used by the S-PLUS help system in assembling its **Category** and **Function** menus.

The following sections describe the steps necessary to generate and distribute an HTML Help-compiled help file from your user-defined help files. We include descriptions and locations of the software you need to download, the elements of the build script, and the components used to create, edit, view, and distribute your customized help files on a Windows platform.

# REQUIRED SOFTWARE AND SCRIPTS

Before you can create a **.chm** file that can be distributed using HTML Help, you need the following two software installations and one distribution directory:

1. **Cygwin** The Cygwin tools are ports of the popular GNU development tools for Microsoft Windows. It is a UNIX environment for Windows that consists of two parts:
   - **cygwin1.dll**, which acts as a UNIX emulation layer providing substantial UNIX API functionality.
   - A collection of tools ported from UNIX which provide UNIX/Linux look and feel.
2. **HTML Help Workshop** This is a help authoring tool that provides an easy-to-use system for creating and managing help projects.
3. **The build_chm_tools directory** This contains the build scripts used for translating the SGML files to HTML and generating the **.chm** and is located in

   **SHOME\help\BuildHelpFiles**

   in your S-PLUS installation.

## Downoading and Running Cygwin

Once you have created and edited your SGML help file, Cygwin is required to run the **build_chm_tools** scripts that translate your SGML files into HTML format, create an index and table of contents (TOC), and fix broken links before a **.chm** file can be created . A free download of Cygwin version 1.3.1 is available from

**www.cygwin.com**

You need approximately 200 MB of disk space for the full Cygwin installation.

| **Note** |
| --- |
| In the **Select install root directory** dialog, set the **Default Text File Type** to **DOS**. The Cygwin default for **Default Text File Type** is UNIX, but you want to use the DOS setting in a Windows environment. |

To use Cygwin from within a `bash` shell, double-click the **Cygwin** icon on your desktop.

To use Cygwin from a DOS Command Prompt (using Windows 2000 or NT):

1. Right-click the **My Computer** icon on your desktop and select **Properties**.
2. Select the **Advanced** tab.
3. Click the **Environment Variables** button.
4. In the **User variables** group, scroll down to find the **path** variable and press **Edit**. If there is no **path** variable, press **New** and enter **path** for the new user variable name.
5. Edit the **path** environment variable so that **cygwin\bin** is included in the path, such as

   **d:\cygwin\bin;%system%;c:\foobar**

This adds **cygwin\bin** to your path so that you can use the Cygwin utilities from a DOS Command. If there are other directories you want to search first, make sure the **cygwin\bin** path is placed at the end of the **path** variable.

## Downloading and Running HTML Help Workshop

After your files have been translated into HTML format and an index and TOC have been generated, the HTML files plus the index and TOC files are compiled into a **.chm** file, a compiled HTML Help project that can be distributed. This required compiler is part of HTML Help Workshop 1.31, which is available for download from

**www.msdn.microsoft.com/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp**

Choose the defaults for the installation.

## The build_chm_tools Directory

This directory contains all the scripts necessary to perform the SGML to HTML translation, create the index and TOC, fix broken links, and create the **.chm** file . You only need to run the **build_helpchm.cyg** script in the **build_chm_tools** directory, which calls other scripts in this “tools” directory.

# CREATING, EDITING, AND DISTRIBUTING A HELP FILE

Now that Cygwin and HTML Help Workshop have been downloaded and you have verified the location of the **build_chm_tools** directory, you are now ready to create a **.chm** file for distribution in S-PLUS 6.1 for Windows. In general, the steps for creating and distributing user-defined help are as follows:

1. In S-PLUS, use the `prompt` function to generate a SGML help file.
2. Edit the SGML file with your Windows editor of choice.
3. Execute the **build_helpchm.cyg** script, either from Cygwin or from a DOS Command Prompt, to create a **.chm** file.
4. Double-click the **.chm** file and view the output.
5. Distribute the **.chm** file by copying it to your **SHOME/cmd** directory.

Repeat steps 2-4 until you are satisfied with the results. If you have a number of files you want to add to your distribution, you can also run this process on a number of help files at once.

To illustrate this process, we create a sample SGML help file, and detail each of the steps involved in creating the **.chm** in the following sections.

## Step 1: Creating the Help File

Let's suppose you create a new S-PLUS function named `myfunc`:

```
> myfunc
function(x)
return(x * 2)
```

Use the `prompt` function to create a template help file named **myfunc.sgml**:

```
> prompt(myfunc)
created file named myfunc.sgml in the current directory
edit the file according to the directions in the file.
```

By default, this file is placed in your S-PLUS working directory. If you have a number of help files to create, use `prompt` for each function.

## Step 2: Editing the Help File

Once the SGML file has been created, you can edit the SGML file in your text editor of choice, such as emacs, Notepad, or Wordpad. Your skeleton help file should contain text similar to the following:

```
<!doctype s-function-doc system "s-function-doc.dtd" [
<!entity % S-OLD "INCLUDE">
]
>
<s-function-doc>
<s-topics>
<s-topic> myfunc </s-topic>
</s-topics>
<s-title>
. . .
```

The first four lines of **myfunc.sgml** are required in all SGML help files for S-PLUS. For this reason, we recommend that you use `prompt` to create a template file rather than write SGML code from scratch.

The `<s-function-doc>` tag begins the contents of the help file; you should see the end tag `</s-function-doc>` at the end of the file. The start and end tags for most fields in an S-PLUS help file are included in the skeleton files that `prompt` creates. If your function does not require a certain field in its help file, you can delete the corresponding tags.

If the text for a given section is particularly long, be sure to start sentences or major phrases on new lines. The SGML parser tends to break when a single line contains too much text. When this occurs, the help file is truncated at the point where the parser breaks and an error message is inserted:

```
Input string too long
```

There is a great deal of information on the meaning and use of the SGML tags in S-PLUS help files, and this is described at length in the section Common Text Formats and the section Contents of Help Files later in this chapter. These sections have been omitted here for brevity.

## Step 3: Executing the Script

Now that **myfunc.sgml** has been edited, you can run the **build_helpchm.cyg** script to translate the SGML to HTML, create an index and TOC, and generate the **.chm** file. You can customize this script to set the name and location of the **.chm** file, the path to the SGML files, the default file that appears in the **.chm**, and other components of the compiled help file.

Let's look at the **build_helpchm.cyg** script and see what each of the variables controls in generating the **.chm** file.

The *first* section of the script allows you to set the name of the target **.chm** file by using the variable `CHM_NAME`:

```
## Set the name of the chm file you are building.  This
## name will be the prefix name of your chm file, such as
## "myhelp.chm"
   CHM_NAME=myhelp
```

The **.chm** suffix is added automatically, so the default name of the compiled file would be **myhelp.chm**.

The *second* section sets the string for the words that appear at the top right and in the title bar of a help file, set by the variable `HEADER`:

```
## Set the string which appears on the right hand side of
## the header gray area for each html page and for the blue
## bar at the top of each window.  This header will be the
## same for both. If there are spaces, make sure the entire
## phrase is surrounded by quotes.
   HEADER="Language Reference for Myhelp"
```

The *third* section sets the target directory for the **.chm** file, using the variable `CHM_DIR`:

```
## Set the path to the directory where you want the chm file
## placed with forward slashes. In addition to the chm file,
## the "help.log" will be placed in this directory.  To view
## the progress of your build in detail, open a cygwin
## window and type "tail -f <path to help.log>/help.log"
   CHM_DIR=D:/myhelpdir
```

This places the compiled HTML Help file **myhelp.chm** in **D:/myhelpdir**.

The *fourth* section sets the directories for the source SGML and target HTML files with the `SGML_DIR` and `HTML_DIR_ROOT` variables, respectively:

```
## Set the path to the directory containing sgml files with
## forward slashes.
## Set the path to the directory containing the __Hhelp
## directory.
## The __Hhelp will contain all the *.html files.
   SGML_DIR=$CHM_DIR/sgml
   HTML_DIR_ROOT=$CHM_DIR
```

For example, when **myfunc.sgml** gets processed at this step, an the source SGML file is in **d:/myhelpdir/sgml/myfunc.sgml**. The generated HTML file will be stored as **d:/myhelpdir/__Hhelp/myfunc.html**. The generated HTML files are used in the creation of the **.chm** file.

The *fifth* section allows you to set the default help file that is displayed in the HTML Help viewer when the **.chm** file is opened. This is set by the `DEFAULTFILE` variable:

```
## Set the name of the *.html file you would like shown as
## the default topic.
   DEFAULTFILE=glm.html
```

Note that the prefix of an **.sgm** file can change when translated into an HTML file. Thus, if you get a compilation error from HTML Help Workshop, make sure the file you specify for the default file is in your HTML directory.

The *sixth* section sets the path to the location of the **build_chm_tools** directory, using the `CHMTOOLS` variable:

```
## Set the path to the directory containing the chm build
## tools with forward slashes.
   CHMTOOLS=D:/modules/moduletools/buildtools/
   build_chm_tools
```

This is the directory included in your S-PLUS installation, and you can move this directory to any available local or network drive as long as you reference it here.

The *seventh* section sets the location of the **cygwin/bin** installation with the `CYGWIN_BIN` variable:

```
## Set the path to cygwin bin using forward slashes.
   CYGWIN_BIN=D:/cygwin/bin
```

Cygwin is used to execute the **build_helpchm.cyg** script (this script), which in turn executes the scripts located in `build_chm_tools`.

The *eighth* section sets the location of the HTML Help Workshop installation and the **hhgen** executable, using the `HELPWORKSHOP_DIR` variable:

```
## Set the path to the directory containing hhgen.exe, the
## *.chm compiler for HTML Help Workshop. Make sure you use
## forward slashes. The path cannot contain spaces.
   HELPWORKSHOP_DIR="D:/Progra~1/HTMLHe~1"
```

The **hhgen** executable is used to create the index file (with a **.hhk** suffix) and the TOC file (**.hhc**) from the generated HTML files stored in `HTML_DIR_ROOT`. The HTML Help Workshop is used to create the **.chm** file from the HTML, index, and TOC files.

The rest of the script is used to pass the variables that are used to generate the **.chm** file, so this completes the descriptions of the various sections found in the **build_helpchm.cyg** script. Note that a log file is generated and is always called **help.log**, and this log file is stored in the same directory as your **.chm** file.

After you are satisfied with changes to this script, save and run it. To execute this script from Cygwin, just `cd` to the directory where **build_helpchm.cyg** is located and type

```
bash build_helpchm.cyg
```

To execute this script from a DOS Command Prompt, type

```
bash build_helpchm.cyg
```

### Step 4: Checking the .chm File

Go to the target directory for your **.chm** file (**D:/myhelpdir** in this example) and double-click the **.chm** file. You should see a file similar to that in Figure 4.1.

Check the various components of your **.chm** file:

- Title of the HTML Help Viewer
- TOC
- Index
- Default help file
- Header string, located in the upper right and the title of the HTML Help Viewer

and any other components set by the variables in the **build_helpchm.cyg** script. If you want to make any changes, iterate steps 2 to 4 until you are satisfied with the results.

### Step 5: Distributing the .chm File

If the **.chm** file is correct, copy it to your **SHOME/cmd** directory, the same location as the compiled help for S-PLUS language functions (**splus.chm**) and the S-PLUS Graphical User Interface (**gui.chm**). When you type

```
>help(myfunc)
```

the help file for `myfunc` should be displayed in your HTML Help Viewer.

### Errors

The **help.log** file (stored in the same directory as the **.chm** file) logs each step executed by the **build_helpchm.cyg** script. Viewing this file in case there are in errors generated from running the scripts or in compiling the **.chm** file.

The rest of this chapter is devoted to the format and syntax of SGML when you are editing the files in step 2.

# COMMON TEXT FORMATS

Table 4.1 lists the most common text formats for S-PLUS SGML files. Note that this list is not exhaustive; in general, there might be many tags to format text in a particular way. We suggest using tags with names that describe the formats themselves. Thus, instead of using <tt> and </tt> for code, we encourage you to use <s-expression> and </s-expression> (or <code> and </code> to save typing).

**Table 4.1:** *Common text formats in S-PLUS SGML files.*

| Format | SGML Tags | Notes |
|---|---|---|
| Fixed-width code font | <s-expression>, </s-expression><br><code>, </code><br><tt>, </tt> | Although <tt> and </tt> still exist in some older help files, please use the other tags instead. |
| Preformatted text | <pre>, </pre> | This is usually for code samples that extend over multiple lines. With <pre>, you are responsible for line breaks and blank space, as SGML does no formatting for you. |
| Italics<br>Emphasis | <it>, </it><br>,  | |
| Bold | <bf>, </bf> | |
| Lists<br>Enumerated lists | <descrip>, </descrip><br><enum>, </enum><br><item>, </item><br><tag>, </tag> | The <descrip> and <enum> tags create the lists. Use <item> and </item>, or <tag> and </tag>, to format the individual list elements. See the nlminb help file for an example. |

**Table 4.1:** *Common text formats in S-PLUS SGML files. (Continued)*

| Format | SGML Tags | Notes |
|---|---|---|
| Line breaks | `<br>` | To include a blank line in a help file, `<br>` must be used twice: once for the text line and once for the blank line. The `<br>` tag is not needed in preformatted sections. |
| Paragraph breaks | `<p>` | This is the recommended tag for including blank lines in a help file. |

# CONTENTS OF HELP FILES

## Descriptions of Fields

Each S-PLUS help file is composed of titled sections that appear in the order listed in Table 4.2. We discuss each of these sections below.

**Table 4.2:** *Titled sections in S-PLUS engine help files. Titles in all capitals appear in a formatted help file. Titles in lowercase letters do not appear in a help file, but the information in those sections do.*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| Topic | The name of the function. | `<s-topics>` `</s-topics>` |
| Title | The title that appears at the top of a formatted help file. | `<s-title>` `</s-title>` |
| DESCRIPTION | A short description of the function. | `<s-description>` `</s-description>` |
| USAGE | The function call with all of its arguments. | `<s-usage>` `</s-usage>` |
| REQUIRED ARGUMENTS | Descriptions of arguments that are required by the function. | `<s-args-required>` `</s-args-required>` |
| OPTIONAL ARGUMENTS | Descriptions of arguments that are optional. | `<s-args-optional>` `</s-args-optional>` |
| VALUE | The return value from the function. | `<s-value>` `</s-value>` |
| SIDE EFFECTS | Side effects from the function. | `<s-side-effects>` `</s-side-effects>` |
| GRAPHICAL INTERACTION | A description of graphical interactions expected of the user. | See below. |
| CLASSES | A description of the classes the function is applicable to, if it is a default method. | See below. |

**Table 4.2:** *Titled sections in S-PLUS engine help files. Titles in all capitals appear in a formatted help file. Titles in lowercase letters do not appear in a help file, but the information in those sections do. (Continued)*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| WARNING | Anything the user should be warned about when using the function. | See below. |
| DETAILS | Descriptions of algorithmic details and implementation issues. | `<s-details>` `</s-details>` |
| BACKGROUND | Background information on the function or method. | See below. |
| NOTE | Any information that does not fit into the above categories. | See below. |
| REFERENCES | Available texts and papers the user can refer to for additional information. | See below. |
| BUGS | Descriptions of known bugs in the function. | See below. |
| SEE ALSO | Links to related S-PLUS functions. | `<s-see>` `</s-see>` |
| EXAMPLES | Coded S-PLUS examples. | `<s-examples>` `</s-examples>` |
| Keywords | A list of keywords that place the help file in the **Contents** topics of the help system. | `<s-keywords>` `</s-keywords>` |

### Topic

The topic section contains the function name wrapped in the tags `<s-topic>` and `</s-topic>`. In the **myfunc.sgml** example, the topic looks like:

```
<s-topics>
<s-topic> myfunc </s-topic>
</s-topics>
```

In help files containing multiple functions, each function name should be wrapped in the <s-topic> tags. For example, the following is an excerpt from the common SGML file for cor and var:

```
<s-topics>
   <s-topic>cor</s-topic>
   <s-topic>var</s-topic>
</s-topics>
```

The topic section is not visible in a formatted help file, but is used to index the file in the help system.

### Title

The title section contains the title that appears at the top of a formatted help file. For example, the title from the coxph SGML file is:

```
<s-title>
Fit Proportional Hazards Regression Model
</s-title>
```

All words in a title should begin in uppercase letters. For clarity, avoid S-PLUS jargon such as function names and class names in the title section. A title should be short enough to fit on one line in the help file.

### DESCRIPTION

This section contains a short description of the function. The description in the coxph SGML file is:

```
<s-description>
Fits a Cox proportional hazards regression model.
Time dependent variables, time dependent strata, multiple
events per subject, and other extensions are incorporated
using the counting process formulation of Andersen and
Gill.
</s-description>
```

### USAGE

This section includes the function call with all of its arguments. You should list optional arguments with the form *name=default*. If a default argument is complicated, use the form *name*=<<see below>> instead and describe the default in the argument's section of the SGML file. Because the angle brackets < and > signify tags in SGML,

however, it is safest to type them as `<` and `>` when tags are not intended. Thus, the form for a complicated default is *name*`=<<see below>>`.

If the help file describes more than one function, the usage for each function should be listed on separate lines. If the argument list for a function is more than one line long, subsequent lines should be indented to one space past the opening parentheses. Use spaces to indent each line instead of tabs. For example, the following is the usage section from the SGML file for `lm`:

```
<s-usage>
<s-old-style-usage>
lm(formula, data=&lt;&lt;see below&gt;&gt;,
   weights=&lt;&lt;see below&gt;&gt;,
   subset=&lt;&lt;see below&gt;&gt;, na.action=na.fail,
   method="qr", model=F, x=F, y=F, contrasts=NULL, ...)
</s-old-style-usage>
</s-usage>
```

The tag `<s-old-style-usage>` preformats the section so that it prints in a fixed-width font. This also causes newlines and blank lines to be recognized in the text. Because of the preformatting, no space is needed around the equals signs for each of the arguments. In addition, please ensure that the width of each line is no more than 60 characters so that the usage section displays nicely in conveniently-sized help windows.

## REQUIRED ARGUMENTS

This section lists each required argument wrapped in the tags `<s-arg name="`*name*`">` and `</s-arg>`. The first word in the description of each argument should not be capitalized. For example, the following excerpt lists the three required arguments from the SGML file for `ifelse`:

```
<s-args-required>
<s-arg name="test">
logical object. Missing values <s-object>(NA)</s-object>
are allowed.
</s-arg>
<s-arg name="yes">
vector containing values to be returned for elements with
<s-expression>test</s-expression> equal to
<s-expression>TRUE</s-expression>.
```

```
</s-arg>
<s-arg name="no">
vector containing values to be returned for elements with
<s-expression>test</s-expression> equal to
<s-expression>FALSE</s-expression>.
</s-arg>
</s-args-required>
```

In descriptions of arguments, you should always state whether exceptional values (`NA`, `NaN`, `Inf`, etc.) are treated specially.

**OPTIONAL ARGUMENTS**

This section lists each optional argument wrapped in the tags `<s-arg name="`*name*`">` and `</s-arg>`. As in the section for required arguments, the first word in each argument's description should not be capitalized, and you should document whether exceptional values are accepted.

**VALUE**

This section describes the return value of the function. The first word in the description should not be capitalized. Often, the description of a function's return value begins with a phrase similar to:

```
a list containing the following components:
```

In this case, you can use the tags `<s-return-component name="`*name*`">` and `</s-return-component>` to format the individual components of the return list. The first word in the description for each return component should not be capitalized. As an illustration, the value section of the `density` SGML file is provided below.

```
<s-value>
a list with two components, <s-expression>x</s-expression>
and <s-expression>y</s-expression>, suitable for giving as
an argument to one of the plotting functions.
<s-return-component name="x">
a vector of <s-expression>n</s-expression> points at which
the density is estimated.
</s-return-component>
<s-return-component name="y">
the density estimate at each <s-expression>x</s-expression>
point.
</s-return-component>
</s-value>
```

You should include arcticles in the uncapitalized sentences of the value section. That is, you should write

```
a list with two components
```

instead of simply

```
list with two components
```

**SIDE EFFECTS**

Side effects of the function (plotting, changing graphics devices, changing session options, etc.) are described in this section. This is also the appropriate place to describe the lack of side effects if a user might expect one. For instance, the fact that many S-PLUS editing functions do not actually change an object can be documented in the side effects sections of their SGML files.

Any function that updates the object `.Random.seed` must include the following message in its side effects section:

> The function *name* causes the creation of the data set `.Random.seed` if it does not already exist. Otherwise, the value of `.Random.seed` is updated.

**GRAPHICAL INTERACTION**

If the user is expected to interact with a graphical display, the interaction is described in this section. Help files that currently contain this section include `burl.tree`, `hist.tree`, and `snip.tree`. The graphical interaction section in S-PLUS SGML files does not have a specific tag. Instead, the tags `<s-section name="`*name*`">` and `</s-section>` are used. For example, the following excerpt is from the `hist.tree` SGML file.

```
<s-section name="GRAPHICAL INTERACTION">
This function checks that the user is in split-screen mode.
A dendrogram of <s-expression>tree</s-expression> is
expected to be visible on the current active screen, and a
graphics input device (for example, a mouse) is required.
Clicking the selection button on a node results in the
additional screens being filled with the information
described above. This process may be repeated any number of
times. Warnings result from selecting leaf nodes. Clicking
the exit button will stop the display process and return the
list described above for the last node selected. See
<s-expression>split.screen</s-expression> for specific
details on graphic input and split-screen mode.
</s-section>
```

**CLASSES**

This section lists the classes the function is applicable to, if it is a default method. Like the graphical interaction section, the classes section in S-PLUS SGML files does not have a specific tag. Instead, the tags `<s-section name="name">` and `</s-section>` are used. For example, the classes section in the `gamma` help file is:

```
<s-section name="CLASSES">
This function will be used as the default method for classes
that do not inherit a specific method for the function or
for the <tt>Math</tt> group of functions. The result will
retain the class and the attributes. If this behavior is
<em>not</em> appropriate, the designer of the class should
provide a method for the function or for the <tt>Math</tt>
group
</s-section>
```

**WARNING**

Anything the user should be warned about when using the function should be described here. The warning section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is the warning section from the `gamma` help file:

```
<s-section name="WARNING">
<s-expression>gamma(x)</s-expression> increases very
rapidly with <s-expression>x</s-expression>. Use
<s-expression>lgamma</s-expression> to avoid overflow.
</s-section>
```

**DETAILS**

Algorithmic details and implementation issues are described in this section. For example, the details section of the `density` help file explains the smoothing algorithm implemented in the function:

```
<s-details>
. . .
These are kernel estimates. For each
<s-expression>x</s-expression> value in the output, the
window is centered on that <s-expression>x</s-expression>
and the heights of the window at each datapoint are summed.
This sum, after a normalization, is the corresponding
<s-expression>y</s-expression> value in the output. Results
are currently computed to single-precision accuracy only.
</s-details>
```

Details that apply to only one argument should be part of the affected argument's description, and not part of the details section.

**BACKGROUND**

Background information on the function or method is described in this section. The text here should be aimed at those with complete ignorance on the subject. The background section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. For example, the background section in the `hclust` help file provides general information on clustering algorithms:

```
<s-section name="BACKGROUND">
Cluster analysis divides datapoints into groups of points
that are "close" to each other. The
<s-expression>hclust</s-expression> function continues to
aggregate groups together until there is just one big
group. If it is necessary to choose the number of groups,
this can be decided subsequently. Other methods (see
<s-expression>kmeans</s-expression>) require that the
number of groups be decided from the start.
<p>
By changing the distance metric and the clustering method,
several different cluster trees can be created from a
single dataset. No one method seems to be useful in all
situations. Single linkage
(<s-expression>"connected"</s-expression>) can work poorly
if two distinct groups have a few "stragglers" between
them.
</s-section>
```

**NOTE**

Anything that does not fit into one of the above categories can be described in this section. The note section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is a note from the `gamma` help file:

```
<s-section name="NOTE">
See <s-expression>family</s-expression> for the family
generating function <s-expression>Gamma</s-expression> used
with the <s-expression>glm</s-expression> and
<s-expression>gam</s-expression> functions. See
<s-expression>GAMMA</s-expression> for the functions
```

```
related to the gamma distribution:
<s-expression>dgamma</s-expression> (density),
<s-expression>pgamma</s-expression> (probability),
<s-expression>qgamma</s-expression> (quantile),
<s-expression>rgamma</s-expression> (sample).
</s-section>
```

## REFERENCES

References for functions are listed alphabetically and should follow *The Chicago Manual of Style*. The format for a book reference is similar to:

> Venables, W.N. and Ripley, B.D. (1999). *Modern Applied Statistics with S-PLUS* (3rd ed.). New York: Springer-Verlag, Inc.

The format for a journal article is similar to:

> Andersen, P. and Gill, R. (1982). Cox's regression model for counting processes, a large sample study. *Annals of Statistics* **10**: 1100-1120.

The references section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is an excerpt from the references in the SGML help file for `density`, which cites a number of different works:

```
<s-section name="REFERENCES">
Silverman, B.W. (1986).
<it>Density Estimation for Statistics and Data
Analysis.</it> London: Chapman and Hall.
<p>
Wegman, E.J. (1972).
Nonparametric probability density estimation.
<it>Technometrics</it> <bf>14</bf>: 533-546.
<p>
Venables, W.N. and Ripley, B.D. (1997)
<it>Modern Applied Statistics with S-PLUS</it> (2nd ed.).
New York: Springer-Verlag.
</s-section>
```

The tag for paragraph breaks `<p>` should be used to separate multiple references. Please try to cite works that are widely available to users.

**BUGS**

In this section, you should document any known bugs a user might encounter while using the function. The bugs section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. For example, the following is a bug from the SGML help file for `subplot`:

```
<s-section name="BUGS">
If you request it to draw outside the figure region,
<s-expression>subplot</s-expression> gets very confused
and typically puts the subplot across the whole region.
</s-section>
```

**SEE ALSO**

This section provides links to related functions. In general, any function you reference in the text of a help file should be linked in this section. The name of each function is wrapped in the tags `<s-function name="filename">` and `</s-function>`; this provides the hyperlinks in the formatted help file. The *filename* is the name of the installed help file, which is the linked function followed by `.sgm`. For example, the links in the SGML file for `subplot` are:

```
<s-see>
<s-function name="symbols.sgm">symbols</s-function>,
<s-function name="locator.sgm">locator</s-function>,
<s-function name="par.sgm">par</s-function>.
</s-see>
```

Functions that rely on the self-doc mechanism for their help files cannot be linked in this section.

Although newlines are not recognized in the see also section of S-PLUS SGML files, spaces are. Thus, be sure to include spaces between each link, even if you type them on separate lines. In the SGML file for `subplot`, two spaces are included at the end of each line, immediately after the `</s-function>` tags.

**EXAMPLES**

The examples in this section should help the user understand the function better. The goal is to provide the user with clear examples that are easily copied and run, either from the commands window or from a script window. Therefore, do not include the S-PLUS prompt character > in your examples and comment any output you include. So that the examples are self-contained, use built-in data sets or create

simple data sets in the code. For clarity, do not abbreviate argument names in your code, and be sure to test your examples before including them in a help file.

The following is an excerpt from the examples section of the `coxph` help file:

```
<s-examples>
<s-example type=text>
# Create the simplest test data set
test1 &lt;- list(time=c(4,3,1,1,2,2,3),
              status=c(1,1,1,0,1,1,0),
              x=c(0,2,1,1,1,0,0),
              sex=c(0,0,0,0,1,1,1))
# Fit a stratified model
coxph(Surv(time, status) ~ x + strata(sex), test1)
. . .
</s-example>
</s-examples>
```

The tag `<s-example type=text>` preformats the examples section so that it prints in a fixed-width font. This also causes newlines and blank lines to be recognized in the text. Thus, you can include spaces between different examples to enhance readability. So that the examples display nicely in conveniently-sized help windows, please ensure that the width of each line is no more than 60 characters.

In your examples, always use the left assignment operator `<-` instead of the underscore _ for assignments. Because the angle bracket `<` signifies a tag in SGML, it is safest to type the left assignment operator using `<` instead. Thus, the operator is `<-` in SGML.

## Keywords

All help files should have keywords listed at the bottom, immediately before the closing tag `</s-function-doc>`. The help system uses keywords to organize the engine functions in the **Contents** tab of the help window; open the **Language Reference** from the S-PLUS menu and click on the **Contents** tab to see this.

Each keyword should be wrapped in the tags `<s-keyword>` and `</s-keyword>`. For example, the keywords section in the SGML file for `coxph` is:

```
<s-keywords>
<s-keyword>models</s-keyword>
<s-keyword>regression</s-keyword>
<s-keyword>survival4</s-keyword>
</s-keywords>
```

This places the `coxph` function in the following **Contents** topics: Statistical Models, Regression, and Survival Analysis. Table 4.3 lists the current keywords and the help topics they map to.

**Table 4.3:** *Current keywords and the **Contents** topics they map to.*

| Keyword | Topic | Keyword | Topic | Keyword | Topic |
|---|---|---|---|---|---|
| `aplot` | Add to Existing Plot | `design` | ANOVA Models | `bootstrap` | Bootstrap Methods |
| `category` | Categorical Data | `character` | Character Data Operations | `cluster` | Clustering |
| `complex` | Complex Numbers | `dplot` | Computations Related to Plotting | `menudata` | Data Menu Functions |
| `wdialogs` | Customizable Dialogs (pre-S-PLUS 4.0) | `wmenus` | Customizable Menus (pre-S-PLUS 4.0) | `attribute` | Data Attributes |
| `data` | Data Directories | `manip` | Data Manipulation | `sysdata` | Data Sets |
| `classes` | Data Types | `chron` | Dates Objects | `debugging` | Debugging Tools |
| `defunct` | Defunct Functions | `deprecated` | Deprecated Functions | `documentation` | Documentation |
| `dynamic` | Dynamic Graphics | `error` | Error Handling | `device` | Graphical Devices |

**Table 4.3:** *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

| Keyword | Topic | Keyword | Topic | Keyword | Topic |
|---|---|---|---|---|---|
| hplot | High-Level Plots | file | Input/Output Files | iplot | Interacting With Plots |
| interface | Interfaces to Other Languages | jackknife | Jackknife Methods | libchron | Library of Chronological Functions |
| libcluster | Library of Clustering Methods | libmaps | Library of Maps | algebra | Linear Algebra |
| list | Lists | loess | Loess Objects | logic | Logical Operators |
| iteration | Looping and Iteration | math | Mathematical Operations | array | Matrices and Arrays |
| methods | Methods and Generic Functions | misc | Miscellaneous | missing | Missing Values |
| mixed | Mixed Effects Models (version 2) | nlme3 | Mixed Effects Models (version 3) | multivariate | Multivariate Techniques |
| nonlinear | Nonlinear Regression | nonparametric | Nonparametric Statistics | optimize | Optimization |
| ode | Ordinary Differential Equations | print | Printing | distribution | Probability Distributions and Random Numbers |
| programming | Programming | qc | Quality Control | regression | Regression |
| tree | Classification and Regression Trees | release | Release Notes | resample | Resampling (Bootstrap, Jackknife, and Permutations) |

**Table 4.3:** *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

| Keyword | Topic | Keyword | Topic | Keyword | Topic |
|---|---|---|---|---|---|
| robust | Robust/ Resistant Techniques | environment | S-PLUS Session Environment | smooth | Smoothing Operations |
| htest | Statistical Inference | menustat | Statistics Menu Functions | models | Statistical Models |
| survival4 | Survival Analysis | ts | Time Series | trellis | Trellis Displays Library |
| guifun | User Interface Programming | utilities | Utilities | DOX | Design of Experiments, Response Surfaces, and Robust Design |
| fracfac | Fractional Factorial Experiments | rsm | Response Surfaces | taguchi | Robust Experimental Design |
| modgarch | GARCH Module for Modeling Time Series Volatility | geostat | Geostatistical Data Analysis | hexbin | Hexagonal Binning |
| lattice | Lattice Data Analysis | pointpattern | Point Pattern Analysis | spatialreg | Spatial Regression |
| spatialstats | Spatial Statistics Module | WAVELETS | Wavelet Analysis of Data, Signals, and Images | swt | Discrete Wavelet Transform Analysis |
| transform1d | 1-D Wavelet and Cosine Transforms | transform2d | 2-D Wavelet and Cosine Transforms | conv | Wavelet Convolutions and Filters |
| cpt | Cosine Packet Analysis | wpt | Wavelet Packet Analysis | crystal | Wavelet Crystals |

**Table 4.3:** *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

| Keyword | Topic | Keyword | Topic | Keyword | Topic |
|---|---|---|---|---|---|
| `molecule` | Wavelet Molecules and Atoms | `wavemake` | Creating Wavelets, Wavelet Packets, and Cosine Packets | `wavelets` | Wavelets Module Functions |
| `wavedata` | Wavelets Module Signals, Images, and Datasets | | | | |

## Special Help Files

Some S-PLUS help files do not fit into the general format described above, and instead require special fields. The two most common types of special help files are those for class objects and data sets. In this section, we briefly list the fields in these types of help files. We do not discuss each field in detail, but refer you to specific help files for more information and example SGML code. As with help files for functions, you can use `prompt` to create template SGML files and delete the tags that are not applicable to your objects.

### SV4 Class Objects

The SGML sections in an SV4 `class.`*`type`* help file are listed in Table 4.4. For more details, see the SGML files for `class.timeSeries`, `class.vector`, and `class.matrix`.

**Table 4.4:** *Titled sections in S-PLUS help files for SV4 class objects.*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| Topic | The name of the object. | `<s-topics>` `</s-topics>` |
| Title | The title that appears at the top of a formatted help file. | `<s-title>` `</s-title>` |
| DESCRIPTION | A short description of the object. | `<s-description>` `</s-description>` |

**Table 4.4:** *Titled sections in S-PLUS help files for SV4 class objects. (Continued)*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| CLASS SLOTS | A list of descriptions for the slots in the object. Each slot can be formatted with the list tags `<s-class-slot name=>` and `</s-class-slot>`. | `<s-slots>` `</s-slots>` |
| EXTENDS | A list of classes the object extends. Each class is formatted with `<s-contains-class name=>` and `</s-contains-class>`. | `<s-contains>` `</s-contains>` |
| DETAILS | Descriptions of implementation issues. | `<s-details>` `</s-details>` |
| NOTE | Any information that does not fit into the above categories. | `<s-section name=>` `</s-section>` |
| REFERENCES | Available texts and papers the user can refer to for additional information. | `<s-section name=>` `</s-section>` |
| SEE ALSO | Links to related S-PLUS functions. The function that creates the object should be included in the links. | `<s-see>` `</s-see>` |
| Keywords | A list of keywords that place the help file in the **Contents** topics of the help system. | `<s-keywords>` `</s-keywords>` |

## Data Sets

The SGML sections in help files for data sets are listed in Table 4.5. For more details, see the SGML files for `solder` and `kyphosis`.

**Table 4.5:** *Titled sections in S-PLUS help files for data sets.*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| Topic | The name of the data object. | `<s-topics>` `</s-topics>` |
| Title | The title that appears at the top of a formatted help file. This should not be the name of the object itself. | `<s-title>` `</s-title>` |

**Table 4.5:** *Titled sections in S-PLUS help files for data sets. (Continued)*

| Section Title | Quick Description | SGML Tags |
|---|---|---|
| SUMMARY | A brief description of the experiment that produced the data. The name of the object should be included in this section. | <s-section name=><br></s-section> |
| DATA DESCRIPTION | A short description of each of the variables in the object. | <s-description><br></s-description> |
| SOURCE | The original references for the data. | <s-section name=><br></s-section> |
| WARNING | Anything the user should be warned about when using the data. | <s-section name=><br></s-section> |
| NOTE | Any information that does not fit into the above categories. | <s-section name=><br></s-section> |
| SEE ALSO | Links to related S-PLUS functions and data sets. | <s-see><br></s-see> |
| EXAMPLES | Coded S-PLUS examples using the data. | <s-examples><br></s-examples> |
| Keywords | A list of keywords that place the help file in the **Contents** topics of the help system. | <s-keywords><br></s-keywords> |

# CONNECT/C++ 5


**Introduction** **94**
Resources 94

**Simple Examples: An Application and a Called Routine** **95**
Creating a Simple Application 95
Example of Calling a C Function Via `.Call` 97

**CONNECT/C++ Class Overview** **101**
Data Object Classes 101
Function Evaluation Classes 101
Client-to-Engine Connection Classes 102
Evaluator Classes 102

**CONNECT/C++ Architectural Features** **104**
`CSPobject` 104
Constructors and Generating Functions 104
Constructing From an Existing Object 105
Assignment Operators 106
Overloading Operators 106
Converting C++ Objects to S-PLUS Objects 107
Subscripting Operators 108
Subscript and Replacement Operations 109
Subscript and Arithmetic Operations 109
Matrix Computations 110
Printing to Standard Output 111
Named Persistent Objects 111
Storage Frames For Unnamed Objects 113

**A Simple S-PLUS Interface** **115**
Creating a Dialog-Based Application 115
Connecting to S-PLUS 116
Evaluating Expressions 117

# INTRODUCTION

CONNECT/C++ is a tool used for interfacing C++ with the S language. It is a convenient tool for integrating the S-PLUS engine inside other programs written in C++, but it can also be used for integrating C++ code into the S-PLUS environment.

To enable communication between the GUI (Graphical User Interface) and S-PLUS, CONNECT/C++ was developed to provide a framework for the S language version 4-based engine used in S-PLUS 6.0 for Windows. In fact, the S-PLUS 6 GUI provides the most comprehensive example of using CONNECT/C++ to integrate the S-PLUS engine with C++ applications. Similarly, C++ developers could create their own GUI to interface with S-PLUS using the same technique.

CONNECT/C++ is a class library providing C++ classes with member functions that operate on S-PLUS objects similar to S methods in the S language. Users can use these classes and their member functions to create and manipulate persistent as well as local S objects.

CONNECT/C++ provides various mechanisms for evaluating S expressions inside a C++ program and module. S-PLUS 6.*x* ships with several examples that illustrate how to use this library. Some of these examples contain pairs of equivalent S and C++ functions that perform the same tasks. The speed of the C++ functions can be many times faster than the S code depending on the code's complexity and the data sizes. The examples are located in the **SHOME\sconnect** directory, where **SHOME** is the location of your S-PLUS installation.

## Resources

For more information on CONNECT/C++, go to **Programs ► S-PLUS 6.1 Professional ► Connect/C++ Library Help**. This HTML file is a guide to the CONNECT/C++ class library for C++ developers, and it discusses how to connect to the S-PLUS engine, how to create data objects, call S-PLUS functions, and evaluate S-PLUS syntax.

# SIMPLE EXAMPLES: AN APPLICATION AND A CALLED ROUTINE

CONNECT/C++ can be used for two distinct purposes: to create C++ applications that can access S-PLUS functionality, and to create C++ functions that can be called via the S-PLUS `.Call` interface. We begin our investigation of CONNECT/C++ with a simple example of each.

## Creating a Simple Application

The CONNECT/C++ application used in this example is a console application that creates two S-PLUS vectors. It then uses S-PLUS to compute a linear model relating the two vectors.

The code begins with the inclusion of **sconnect.h**, the CONNECT/C++ library which all CONNECT/C++ code must reference at the start. It then declares a global S-PLUS connection object, with the CONNECT/C++ class `CSPengineConnect`, before beginning the main application function. The `CSPengineConnect` class generates a connection between the client application and S-PLUS, allowing you to create S-PLUS objects in the permanent frame, notifying you when the databases are attached or detached to the client, and evaluating S language expressions. Here's what the code looks like so far:

```
#include "sconnect.h"

// A global connection object
CSPengineConnect g_engineConnect;

int main(int argc, char* argv[])
{
```

The first step in the main function is to create the actual connection object, which opens a connection to S-PLUS:

```
// Create the connection to S-PLUS
g_engineConnect.Create( argc, argv);
```

We then create the variables `x` and `y` to use in the regression. The CONNECT/C++ class `CSPnumeric` is used to store S-PLUS numeric vectors. The `CSPnumeric` class is one of many in CONNECT/C++ that are used to represent S-PLUS objects within C++. Similar classes

exist for most of the standard atomic objects in S-PLUS (see Table 5.1). The `Create` method creates instances of the class; the `Assign` method assigns the class to an S-PLUS database:

```
// Create S object with name "x" in the current database.
// Same as x<-1:10 at the command line.
CSPnumeric sx;
sx.Create("1:10","x");

// Squaring sx, which is the same as S expression
// sy <- x*x in a local frame, but here we set it to local
// C++ variable sy.

CSPnumeric sy = sx * sx;

// Assign the result as S object with name "y" in the
// current database.
sy.Assign("y");
```

Finally, we fit the linear model, passing the appropriate call to S-PLUS via the CONNECT/C++ method `SyncParseEval`:

```
  // Evaluate z<-lm(y~x)
  g_engineConnect.SyncParseEval("z<-lm(y~x)");

  return 1;
}
```

The complete code for this example is in the directory **SHOME\samples\spllm**, and the C++ code is in the file **spllm.cxx**.

Open a Windows Command Prompt or MS-DOS window (depending upon your operating system) and compile and run the application as follows:

1. Change the current directory to the directory containing the code:

   ```
   cd SHOME\samples\spllm
   ```

   where **SHOME** is your installation directory.

2. Build the program:

   ```
   msdev spllm.dsp /make
   ```

3. Check the PATH environment variable to make sure it includes **%SHOME%\cmd**. It should already be added by the S-PLUS installer. If not, you must add it before running the next step.
4. Run the program:

```
splm.exe S_PROJ=.
```

To verify the results, start the S-PLUS console version in the same directory:

```
sqpe.exe S_PROJ=.
S-PLUS : Copyright (c) 1988, 2002 Insightful Corp.
S : Copyright Lucent Technologies, Inc.
Version 6.1.2 Release 1 for Microsoft Windows : 2002
Working data will be in E:/programs/splus6/users/rich
```

and look at the objects x, y, and z:

```
> x
[1]  1  2  3  4  5  6  7  8  9  10
> y
[1]  1  4  9  16  25  36  49  64  81  100
> z
Call:
lm(formula = y ~ x)

Coefficients:
 (Intercept)  x
         -22 11

Degrees of freedom: 10 total; 8 residual
Residual standard error: 8.124038
```

## Example of Calling a C Function Via .Call

The Gauss-Seidel method is a familiar technique for solving systems of linear equations. The algorithm is straightforward and easy to implement in S-PLUS:

```
gaussSeidel<-
# gaussSeidel solves a linear system using Gauss-Seidel
# iterative method.
# REQUIRED ARGUMENTS:
#      A and b are numeric matrix and vector respectively.
```

```
# VALUE:
#      a vector x, solution of A x = b
#
# Usage:
#  A<-matrix(rnorm(100),nrow=10)
#  diag(A)<-seq(ncol(A),ncol(A)) #Make it diagonally
#                                # dominant
#  b<-rnorm(ncol(A))
#  sys.time({x1<-gaussSeidel(A,b)})
function(A,b)
{
  # Hard-coded relative tolerance and max iterations
  tol<-1.0e-4
  maxItr<-1e4

  # Validating
  A <- as.matrix(A)
  b <- as.numeric(b)
  if(nrow(A)!=ncol(A) || ncol(A)!=length(b))
     stop("nrow(A)!=ncol(A) || ncol(A)!=length(b)")

  # Begin Gauss-Seidel step
  x<-b
  for(k in 1:maxItr)
  {
     xOld<-x
     for(i in 1:nrow(A))
     {
        s<- A[i,i]*x[i]
        for(j in 1:ncol(A))
           s <- s - A[i,j]*x[j]
        x[i] <- (b[i]+s)/A[i,i]
     }
     # Check convergence; continue if necessary
     if(max(abs((x-xOld)/x)) < tol)
        return(x);
  }
  warning("Solution does not converge\n")
  return(x)
}
```

This code, which involves a nested loop, could be made more efficient, but the intention is to illustrate the Gauss-Seidel iteration in its most familiar form. An example including the implementation of CONNECT/C++ is shown below, and notice that by using the classes and methods of CONNECT/C++, this code closely resembles the equivalent computation in S-PLUS.

The code begins by including the **sconnect.h** header file to give us access to the CONNECT/C++ library. Next, it includes the header file required for the Gauss-Seidel code itself:

```
# include "sconnect.h"
# include "gausssdl.h"
```

We then declare the `gaussSeidel` object as an object of class `s_object`, as required by the `.Call` interface:

```
s_object* gaussSeidel(s_object* ps_A, s_object* ps_b)
```

As is typical for S-PLUS code, we declare the `S_EVALUATOR` and then embed the implementation in a `try-catch` block. Within the `try` block, the tolerances are hard-coded. We then construct the C++ equivalents to the S-PLUS objects `A` and `b`:

```
{
  S_EVALUATOR
  try
  {
    // Hard-coded relative tolerance and max iterations
    double tol =1e-4;
    long maxItr = 1000;

    // Constructing and validating C++ objects
    CSPnumericMatrix A(ps_A);
    CSPnumeric b(ps_b);
    if(A.nrow()!=A.ncol() || A.ncol()!=b.length())
      PROBLEM "A.nrow()!=A.ncol() || A.ncol()!=b.length()"
       ERROR;
```

The actual Gauss-Seidel step follows:

```
    // Begin Gauss-Seidel step
    CSPnumeric x=b;
    for(long k =1; k<= maxItr; k++)
```

```
    {
      CSPnumeric xOld = x;
      for(long i= 1; i <= A.nrow(); i++)
      {
        double s = A(i,i) * x(i);
        for(long j = 1; j <= A.ncol(); j++)
          s  = s - A(i,j) * x(j);
        x(i) = (b(i)+s)/A(i,i);
      }
      // Check convergence; continue if necessary
      if(Max(abs((x-xOld)/x)) < tol)
          return(x);
    }
    PROBLEM "Solution does not converge" WARN;
    return(x);
  }
  catch(...)
  {
  }
  return(blt_in_NULL); // return the built-in NULL object
}
```

CONNECT/C++ called via `.Call` runs considerably faster than the S-PLUS code equivalent. The following is a comparison for a 100 column by 100 row matrix `A` using a Pentium III with 512MB of RAM:

```
> A<-matrix(rnorm(10000),nrow=100); diag(A)<-seq(ncol(A),
+ ncol(A)) # Make it diagonally dominant
> b<-rnorm(100);
> sys.time({x1<-gaussSeidel(A,b)})
[1] 19.328 19.354
> sys.time({x2<-.Call('gaussSeidel',A,b)})
[1] 0.07 0.07
```

The CONNECT/C++ version ran over 250 times faster than the pure S-PLUS version!

# CONNECT/C++ CLASS OVERVIEW

The class library provides a set of classes that can be used to create and manipulate persistent data objects, run S-PLUS functions, parse and evaluate S-PLUS expressions, and receive output and notification when objects are changed or when databases are attached and detached.

The following sections provide an overview of specific categories of classes used to accomplish these operations.

## Data Object Classes

Data object classes provide methods to create and operate on arrays, matrices, and vectors. To use these classes to create a data object, simply call the object constructor or call the `Create()` method. For a persistent object, specify the name of the object and an S language expression you want to parse, evaluate, and assign the result in order to initialize it with data. Alternatively, a data object can be constructed using a form of the constructor that takes an optional S language expression as an argument. This is useful if named (persistent) objects are not required, but intialization is required. Once the object is created, methods can be used to operate on the object.

To receive notification in a client application when a data object changes, create a new class in the client application derived from the appropriate base class and override the virtual methods for handling object notification. When a named object is modified or removed, those virtual methods in the client are called.

## Function Evaluation Classes

The `CSPcall` class allows S-PLUS functions to be evaluated with arguments passed to the function. Arguments are any `S_object` as well as objects derived from `CSPobject`, which may include data objects and other S-PLUS objects. Results are returned as a `CSPobject` to the client. To use this class, simply call the object constructor with the name of the function to run and any arguments you wish to pass from the client to the function.

## Client-to-Engine Connection Classes

The `CSPengineConnect` class creates a connection between the client and the S-PLUS engine. This connection permits creation of objects in the permanent frame, creation of persistent unnamed objects outside of `.Call` routines, notification in the client when databases are attached or detached, output routing to the client, and evaluation of S language expressions.

To use `CSPengineConnect`, create a new class derived from `CSPengineConnect` in the client, override the virtual methods for receiving database attach/detach notification, and output notification, and add a member variable to the client application class object to record a reference to a single instance of this derived class.

Use of the `CSPengineConnect` class is only necessary when one or more of the following features is desired in the client program:

- Integrate S+ engine DLLs with another application (client).
- Notification in the client when databases are attached or detached and when changes are made in persistent objects.
- Output redirected to the client.

For more information on using `CSPengineConnect`, please see the section on this class by going to **Programs ▶ S-PLUS 6.1 Professional ▶ Connect/C++ Library Help**.

## Evaluator Classes

The `CSPevaluator` class manages memory resources, errors, the top-evaluation frame, and a set of local evaluation frames. Although it is optional, instantiating an object of `CSPevaluator` class at the top of a `try` block can speed up the code, and the corresponding `catch` block receives an exception error when an unexpected error occurs in the S-PLUS engine.

To use `CSPevaluator`, create an instance of this class at the top of a `try` block as shown below:

```
double minValue = 0;
try
{
      // Open top-level-evalutor (frame 1) if it is closed
      CSPevaluator sEvaluator;
      CSPnumeric myNumeric = sEvaluator.eval("1:10");
      minValue = myNumeric.Min(); //minValue = 1
```

```
} // Close top-level evaluator when sEvaluator is out of
  // scope
catch(...)
{
      // Unexpected error occurred in the engine
}
```

For more information on using `CSPevaluator`, please see the section on this class by going to **Programs ▶ S-PLUS 6.1 Professional ▶ Connect/C++ Library Help**.

# CONNECT/C++ ARCHITECTURAL FEATURES

The following sections describe the basic architectural featues in the class library and some of the specific programming features available in the library that make it possible to perform S-PLUS operations efficiently in client programs and modules written in C++. Classes and methods discussed in this section are fully documented in the reference sections for the classes in the online help for CONNECT/C++.

## CSPobject

`CSPobject` is the base class of most of the classes that represent S-PLUS classes. It provides common functionality to its derived classes, and its most important data member is:

```
s_object* CSPobject::m_ps_object
```

A class that represents an S-PLUS class inherits `m_ps_object` because `CSPobject` is its base class. As a smart pointer, a derived class of `CSPobject` provides safer methods to manipulate the data pointed by `m_ps_object` as compared to using global C functions. For example, the constructor, the destructor, and the assignment operators automatically increment and decrement reference counts whenever appropriate to provide the same data sharing mechanism as that of the SV4 language.

All `CSPobject`-derived classes have a method called `IsValid()` which allows you to test whether the member `m_ps_object` is valid or not.

## Constructors and Generating Functions

Often, S generating functions are more convenient than the S method `new`. Similarly, constructors of CONNECT/C++ classes can provide the same convenience. They have the following form:

```
CSPclass::CSPclass(const char* pszExpression);
    // pszExpression is a string representing valid S code.
```

where `class` is a `CSPobject`-derived object.

This form of the object constructor parses and evaluates `pszExpression` and uses the resultant S-PLUS object as its value. Normally, `pszExpression` should contain an S-PLUS expression that calls to an appropriate generating function. However, it works for any S-PLUS expression that returns a valid S-PLUS object, and the

constructor automatically coerces the returned object to the class that it represents. It increments the reference count upon completion, as well. In case of errors, the constructor throws an exception in the client application.

For example:

```
CSPevaluator s;

CSPinteger x("1:4");              // x<-1:4
CSPnumeric y("fuel.frame[,1]");   // y<-as(fuel.frame[,1],
                                  //   'numeric')
CSPnumeric z("new('numeric')");   // z<- new('numeric')
CSPmatrix A("matrix(1:4, nrow=2)"); // A<-matrix(1:4,
                                    //     nrow=2)
CSPmatrix B("1:4");               // B<-as(1:4,'matrix')

// Do something with x,y,z,A, and B
```

## Constructing From an Existing Object

You can construct new objects from existing objects using one of the following forms:

```
CSPclass::CSPclass(const CSPclass& sObject);//copy
                                            //constructor
CSPclass::CSPclass(s_object* ps_object);    //construct
                                            //from s_object
```

where `class` is a `CSPobject`-derived object.

The copy constructor of a CONNECT/C++ class behaves like an S-PLUS assignment operator when the S-PLUS object name is first used. They both share the same data with the object names used to construct them. However, for the CONNECT/C++ classes, sharing is not possible if the classes are incompatible. It increments the reference count upon completion.

An example of creating new objects from existing objects follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric u(x);     // u<-x # u shares data with x
CSPmatrix A(x); // A<-as(x,'matrix') # A shares data with x
```

```
CSPcharacter v(x);   // v<-as(x,'character') # no sharing

s_object* ps_object = x.GetPtr();//Get pointer to s_object*
CSPnumeric U(ps_object);         // U shares data with x
CSPmatrix a(ps_object);          // a shares data with x
```

## Assignment Operators

The assignment operator of an CONNECT/C++ class behaves like an S-PLUS assignment operator when the S-PLUS object name is already used. However, the left-hand-side object of the operator = is an existing and valid object. The assignment operator decrements the reference count on the old object and increments the reference count on the new object before swapping the two object pointers:

```
CSPclass& CSPclass::operator=(const CSPclass& sObject);
```

where `class` is a `CSPobject`-derived object.

An example of the assignment operator follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric u = x;   // u<-new('numeric'); u<-x # u shares
                    // data with x
CSPmatrix A = x;   // A<-new('matrix'); A<-as(x,'matrix')
                   // # no sharing

CSPnumeric y;  // y<-new("numeric")
u = y;       // u<-y # u switches to share data with y
A = y;//A<-as(y,'matrix') # A switches to share data with y
```

## Overloading Operators

CONNECT/C++ contains some useful overloading operators such as +, -, * and /. These operators perform element-by-element operations in the same way as in the S language. However, for the matrix class, the * operator is different. The operator for `CSPmatrix` is a real matrix multiplication operator equivalent to the S `%*%` operator.

```
CSPclass& CSPclass::operator+(const CSPclass& sObject);
CSPclass& CSPclass::operator-(const CSPclass& sObject);
CSPclass& CSPclass::operator*(const CSPclass& sObject);
CSPclass& CSPclass::operator/(const CSPclass& sObject);
```

where `class` is a `CSPobject`-derived object.

An example using the CSPmatrix follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric y ("4:1"); // y<-4:1
y = y+x*x;    // y<-y+x*x

CSPmatrix A("matrix(1:4,nrow=2)");//A <- matrix(1:4,nrow=2)
CSPmatrix B("matrix(4:1,nrow=2)");//B <- matrix(4:1,nrow=2)
CSPmatrix D = A*A + B*B;          //D <- A %*% A + B %*% B
```

## Converting C++ Objects to S-PLUS Objects

Objects derived from class CSPobject are C++ representations of S-PLUS objects; within S-PLUS, S-PLUS objects are represented as C objects of type s_object*. Sometimes, an application needs to access the s_object* directly. For example, the arguments and the return value of all .Call interfaces must be of type s_object*.

The CSPobject class provides a convenient way to automatically convert to s_object*. Simply use a CSPobject wherever a s_object* is required. It automatically invokes a conversion operator that returns the s_object* as appropriate.

```
s_object* CSPobject::operator*();
s_object* CSPobject::operator&();
```

For example:

```
s_object* myCall()
{
        CSPnumeric x("1:10");
        return x;
}
s_object *pReturn = myCall();
```

The return statement, return x, first typecasts x to type s_object*. This invokes the conversion operator s_object *() of the CSPnumeric class (derived from CSPobject) which ensures that the destructor of x does not delete the object, even if the reference count drops to zero.

## Subscripting Operators

CONNECT/C++ contains some useful overloading subscripting operators `()` for the derived classes of `CSPvector` and `CSParray` such as `CSPnumeric` and `CSPmatrix`. The proxy class of the returned object provides supports for read/write and mixed-mode operations:

```
const double CSPnumeric::operator()(long lIndex); const
      // Fortran style indexing starting from index 1
      // rvalue only

CSPproxy CSPnumeric::operator()(long lIndex);
      // Fortran style indexing and ordering
      // lvalue and rvalue
```

An example using the subscripting operators:

```
CSPevaluator s;

CSPnumeric x("c(0.1, 0.2, 0.8, 0.9)"); // x<- c(0.1, 0.2,
                                       //       0.8, 0.9)
double d = x(1);   // d <-x[1] # d is 0.1
d = d + x(2);      // d<- d+x[1] # d is 0.3
double e = (long) x(1);  // e<-as.integer(x[2]) # e is 0
long n = x(1);           // n <-as.integer(x[1]) # n is 0
n = n + x(2);   // n <- n+as.integer(x[2]) # n is still 0
```

The following is another example using the subscripting operator for a matrix:

```
CSPevaluator s;

CSPmatrix A("matrix(c(0.1, 0.2, 0.8, 0.9),  2)");
       // A<- matrix(c(0.1, 0.2, 0.8, 0.9),  2)

double d = A(1,1);         // d <-A[1,1] # d is 0.1
d = d + A(2,1);            // d<- d+A[2,1] # d is 0.3
long e = (long) A(2,1); // e<-as.integer(A[2,1]) # e is 0
long n = A(1,1);       // n <-as.integer(A[1,1]) # n is 0
n = n + A(2,1); //n <- n+as.integer(A[2,1]) # n is still 0
```

## Subscript and Replacement Operations

If a subscript operator of a `CSPobject`-derived class returns an `lvalue` object of `CSPproxy`, the operation involves replacing an element of the S-PLUS object. Since writing data is not possible for a shared S-PLUS object, `CSPproxy` must determine whether to copy data before replacing its elements. This action occurs in one of its overloaded assignment operations:

```
CSPproxy& CSPproxy::operator=(long);
CSPproxy& CSPproxy::operator=(double);
CSPproxy& CSPproxy::operator=(const CSPproxy&);
```

For example:

```
CSPevaluator s;

CSPnumeric x("1:4");        // x<- 1:4
x(1) = 0.0;                 // x[1]<- 0 # x is not share,
                            // simply set x[1] to 0.0
x(2) = x(1);       // x[2]<- x[1] # x is not share, simply
                   // set x[2] to 0.0
CSPnumeric y(x);   // y<- x # y shares data with x
y(1)= 10.0;        // y[1]<- 10 #copy and replace:
                   // y[1] is 10 and x[1] is 0
```

## Subscript and Arithmetic Operations

Some overloaded operators are available to support mixed-mode arithmetic operations involving subscripting objects of classes derived from `CSPobject`. These operators, `+`, `-`, `*` and `/`, perform mixed-mode operations following the same rules as S-PLUS:

```
long CSPproxy::operator+(long)
double CSPproxy::operator+(double)
...
```

An example using the arithmetic operators:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<- 1:4
CSPnumeric y(x);     // y<- x # y shares data with x
// A <- matrix(1:4,nrow=2)
CSPmatrix A("matrix(1:4,nrow=2)");
```

```
// e <- A[1,1] + A[1,2]
double e = A(1,1)+A(1,2);
// A[1,2] <- e*(A[1,1]+A[2,1])
A(1,2) = e*(A(1,1)+A(2,1));
// A[2,2] <- x[1]*A[1,1]+y[2]*A[2,1]
A(2,2) = x(1)*A(1,1)+y(2)*A(2,1);

// X<-array(1:16, c(2,2,2,2))
CSParray X("array(1:16, c(2,2,2,2))");
// X[1,1,1,1] <- X[2,1,1,1]+e;
X(1,1,1,1) = X(2,1,1,1) + e;
// X[2,1,1,1] <- y[1] - X[2,1,1,1];
X(2,1,1,1) = y(1) - X(2,1,1,1);
// X[1,2,1,1] = A[1,1] * X[2,1,1,1];
X(1,2,1,1) = A(1,1) * X(2,1,1,1);
```

## Matrix Computations

Some overloaded functions are available for matrix computations. These computations are multi-threaded on some platforms (currently Windows 2000, NT, and XP on Intel multi-processor machines).

```
double CSPmatrix::ConditionNumber(void);
CSPmatrix SPL_Multiply(const CSPmatrix& A,
      const CSPmatrix& B);
CSPnumeric SPL_Multiply(const CSPmatrix& A,
      const CSPnumeric& x);
...
```

For example:

```
CSPevaluator s;

CSPmatrix A("matrix(5:8, nrow=2)");
 // A<- matrix(5:8, nrow=2)
CSPmatrix B(A);                    // B<- A
CSPmatrix D = SPL_Multiply(A, B); // D<-A %*% B

CSPnumeric x("1:2");                // x<- rnorm(2)
CSPnumeric y = SPL_Multiply(A, x); // y<- A %*% x
```

## Printing to Standard Output

You can use the following CONNECT/C++ method to print to the S-PLUS standard output stream:

```
void CSPobject::Print(void);
```

For example:

```
CSPevaluator s;

CSPcharacter message("'hello'");  //message <- 'hello'
message.Print();                  //print(message)

CSPmatrix M("matrix(1:4,nrow=2)");//M<-matrix(1:4, nrow=2)
M.Print();                        //print(M)
```

## Named Persistent Objects

All `CSPobject`-derived objects are placeholders for an `s_object` that exists in the engine. So, this C++ object can reference an `s_object` or none at all, depending on whether the member `s_object` pointer points to a valid `s_object`. All `CSPobject`-derived classes have a method called `IsValid()` which allows you to test whether it is pointing to a valid `s_object` or not.

All named objects are created in a permanent frame associated with an S-PLUS database, and are thus persistent between calls and between sessions in the S engine. When you create a new `CSPobject` in your client program, a new `s_object` is created in the S engine. When you delete this `CSPobject`, the `s_object` is also released in the engine. However, when you execute S-PLUS expressions to remove the `s_object` that your `CSPobject` points to, such as by using `rm(myObject)`, or you call the `Remove()` method on the object, the `CSPobject` is not deleted in your client. The `OnRemove()` method of the `CSPobject` in your client is called and the base class version of this method "disconnects" your `CSPobject` from the now released `s_object` by setting the member `s_object` pointer to `NULL`. After this event, calling `IsValid()` on the `CSPobject` returns `FALSE`.

Deleting the `CSPobject` in your client program does not automatically remove the permanent frame `s_object` in the S-PLUS engine that this `CSPobject` refers to. You must call the method `Remove()` to remove the `s_object` from the engine.

You can create named objects using the `Create()` method of the various object classes derived from `CSPobject`, such as `CSPnumeric`. Whenever these objects are modified, the `OnModify()` method is

called in your client program. Whenever these objects are removed, the OnRemove() method is called in your client program. Only named objects support this kind of client program notification.

To create a named object in your client, first derive a new class from the appropriate CSPobject-derived class, such as CSPnumeric. Then, construct an instance of this derived class using the constructor, then call the Create() method to specify the name you wish to give the object. It is important to derive a new class from the CSPobject-derived class instead of just using the base class directly in your client because the OnModify() and OnRemove() methods are virtual and must be overridden in your derived class in the client in order to be notified when these events occur.

A CSPobject can be modified in one of two ways. It can be modified in the client program by using the operators available for the object to assign and operate on the elements of the object. When this kind of modification is done, it is necessary to call the Commit() method on the object to commit it to the S-PLUS engine before any changes to the object are reflected in the persistent s_object that is referenced by the object in the client.

Another way it can be modified is by evaluating S-PLUS expressions, such as by using CSPengineConnect::SyncParseEval(). When this kind of modification is done, it is not necessary to call Commit() on the object, as the s_object is automatically updated by the S-PLUS engine. For both kinds of modification, the OnModify() method of the CSPobject is called in the client program. It is important to call the base class OnModify() in your override of OnModify(). This allows the base class to update the member s_object pointer to point to the newly modified s_object in the engine.

The s_object member of a CSPobject can be removed (invalidated) in one of two ways:

1. It can be removed in the client program by calling the Remove() method on the CSPobject. This method removes the s_object from the permanent frame and triggers a call to the OnRemove() method of the CSPobject in the client program. The base class version of OnRemove(), which should be called at the end of the overridden version in the client, releases the member s_object from the CSPobject.

2. It can be removed by evaluating S-PLUS expressions, such as by calling `CSPengineConnect::SyncParseEval()`. This also triggers a call to the `OnRemove()` method of the `CSPobject` in the client program.

For examples of using `CSPobject`-derived classes in a client program and responding to `OnModify()` and `OnRemove()` notifications, see the example C++ client program called **SSC** located in **SHOME\samples\SSC** in the subdirectory.

## Storage Frames For Unnamed Objects

Normally, when you create an unnamed `CSPobject` in a client routine that you call via `.Call`, the `s_object` corresponding to this `CSPobject` is "alive" or is valid until the routine ends and scope changes out of the routine.

If you create an unnamed `CSPobject` when the S-PLUS evaluator is not open, the `s_object` corresponding to this `CSPobject` may not be valid. For most client applications, this is usually inadequate. Therefore, you need to do the following to ensure that an unnamed `CSPobject` created in a client application does not get released until the end of the client routine:

- Create an instance of a `CSPevaluator` at the top of the scope "`{`."
- Create and use any unnamed `CSPobject`-derived objects in the client.

For example:

```
{
        CSPevaluator s;
        CSPnumeric x("1:10");
        ...
}
```

For named objects, you do not have to use the above approach: simply create named `CSPobject`-derived objects using the constructor and a call to `CSPobject::Create()`. For further information, see the online help for the classes `CSPengineConnect::OpenTopLevelEval()`, `CSPengineConnect::CloseTopLevelEval()`, and the `Create()` method for the object type to be created.

**Table 5.1:** *CONNECT/C++ classes and their S-PLUS counterparts.*

| S-PLUS Class | CONNECT/C++ Class | Example |
|---|---|---|
| any | CSPobject | CSPobject x("2") |
| numeric | CSPnumeric | CSPnumeric x("2.32") |
| integer | CSPinteger | CSPinteger x("2") |
| logical | CSPlogical | CSPlogical x("c(T,F") |
| character | CSPcharacter | CSPcharacter("abcd") |
| named | CSPnamed | CSPnamed("c(a=1,b=2, d=3)") |
| matrix | CSPmatrix<br>CSPnumericMatrix<br>CSPcharacterMatrix | CSPmatrix A("matrix(1:4,2)")<br>CSPnumericMatrix A("matrix(rnorm(12,6")<br>CSPcharacterMatrix A("matrix(letters[1:12],6)") |
| array | CSParray | CSParray B(“array(1:8,c(2,2,2))" |
| list | CSPlist | CSPlist("list(1:2,6:700)") |
| function | CSPfunction | CSPfunction ("function(x) x^2") |
| call | CSPcall | CSPcall("lm") |

# A SIMPLE S-PLUS INTERFACE

In this section, we build a small client application to accept S-PLUS expressions, send these expressions to S-PLUS for evaluation, then return the output from S-PLUS. As part of this process, we use numerous features of CONNECT/C++.

| **Warning** |
| --- |
| Most of this example was automatically generated using Visual C++, and it uses Microsoft Foundation Classes (MFC). If you are not familiar with MFC, it is assumed you can ignore the uses of MFC where they occur. |

## Creating a Dialog-Based Application

To keep our application as simple as possible, we create it as a dialog-based application in Visual C++. The basic interface is simple to create:

1. Open Visual C++.
2. Choose **File ▶ New**.
3. From the **Projects** tab, select **MFCAppWizard** (**exe**).
4. Type a project name, such as **spint**, in the **Project name** text field.
5. Click **OK**. The **MFCAppWizard** appears.
6. Under **What type of application would you like to create**, select **Dialog-based**. Click **Next**.
7. Choose the features you want included in your application. Accept the defaults by clicking **Next**.
8. Accept the defaults in the displayed choices by clicking **Next**.
9. Click **Finish** and then **OK** to create your application skeleton.

The application skeleton consists of a project file plus 11 files that you can view by selecting the **File View** tab in the left pane of Visual C++; the file **ReadMe.txt** contains a brief description of the other files in the skeleton. The most important for our purposes are the files **spint.cpp**, **spintDlg.cpp**, **spint.rc**, **spint.h**, and **spintDlg.h**.

10. The file **spint.rc** is open for editing when the skeleton is first created. Use the **Controls** palette to add two static text fields and two edit fields to the dialog.
11. Rename the static text fields to read **S-PLUS Commands** and **S-PLUS Output**, respectively. An edit field should follow each static text field, and the edit field following **S-PLUS Output** should be set to read-only via the control's **Properties** dialog (available by right-clicking the control).
12. Rename the **OK** button to read **Run Commands**. The completed dialog should look like the following:

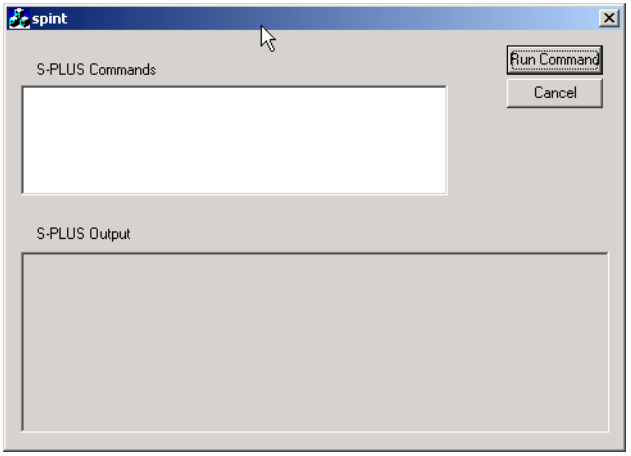



**Figure 5.1:** *Our simple S-PLUS interface.*

## Connecting to S-PLUS

To establish a connection between S-PLUS and our dialog, we need to edit the main source file for our application, **spint.cpp**, and the main header file, **spint.h**. The change to **spint.h** is simple—at the very top of the file, include the header file **sconnect.h** immediately after the resource header file:

```
// spint.h : main header file for the SPINT application
//

#if !defined(AFX_SPINT_H__F2C4C05F_6855_40FB_B41D_1D50CC25A174__INCLUDED_)
#define AFX_SPINT_H__F2C4C05F_6855_40FB_B41D_1D50CC25A174__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"    // main symbols
#include "sconnect.h"    // Connect/C++ library
```

The change to **spint.cpp** is more substantive, but still straightforward. In the section of the code titled `CSpintApp initialization`, add the following code immediately before the line reading `CSpintDlg dlg;`:

```
// Create and connect to S+ engine
int argc =1;
char *argv[1];
argv[0]="spint";

m_spconn.Create(argc, argv);
```

## Evaluating Expressions

We now have an interface and a connection to the engine. All that remains is to define the code that actually reads the input from the **S-PLUS Commands** edit field and writes the output to the **S-PLUS Output** edit field. The following function, added at the end of the file **spintDlg.cpp**, does what we want: it takes the input, reads it into a `CString` object, sends it to S-PLUS for evaluation, returns the output as an array of character strings, then takes the first output string and puts it into the output field:

```
void CSpintDlg::OnRunCommand()
{
  CWnd* pEdit = GetDlgItem(IDC_EDIT1);
  CString Commands1;
  pEdit->GetWindowText(Commands1);
  CSPevaluator sEvaluator;
  CSPobject returnVals=sEvaluator.Eval(Commands1);
  CSPcharacter outputText=returnVals.Deparse();
```

```
    CString outputText2=outputText[0];
    CWnd* pEdit2 = GetDlgItem(IDC_EDIT3);
    pEdit2->SetWindowText(outputText2);
}
```

This implementation, however, has one significant limitation: we get only the first string in the array of strings that forms the return value. What this means in general is that all the output pane shows us is the first line of output.

A solution to this problem is provided by defining a new class derived from `CSPengineConnect` that includes a new method for `OnOutput`. You can view this solution by exploring the code in **SHOME\samples\ssc**, a more sophisticated "simple" interface to S-PLUS.

# INDEX


## Symbols

.Call interface 95, 97, 99, 100, 102, 107, 113
.chm file 64, 66, 67, 68, 70, 71, 72, 73
.JavaField function 22, 25, 44
.JavaMethod function 25, 26, 29, 44

## Numerics

1000s Separator field 61

## A

Analytic Server, S-PLUS 20, 24, 26
arithmetic operators 109
ASCII file formats 61
Assign 96
assignment operators 104, 105, 106

## B

build_chm_tools 66, 67, 68, 71, 72
build_helpchm.cyg 67, 68, 70, 72, 73

## C

C++
    *See* CONNECT/C++
catch block 102
characters
    allowed in variable names 57
    not allowed in variable names 58
character sets 6, 52, 53, 57
    ISO 8859-1 6, 57, 58
class descriptors 22
classes
    CONNECT/C++ 94, 95, 99, 101, 104, 105, 108, 109, 111, 113, 114
    connection 102
    data object 101
    evaluator 102, 103
    function evaluation 101
class files, Java 26, 27, 42
class library 94, 101, 104
classpath, Java 26, 27, 28
C locale 53, 55, 56
collating 52
collation sequences 6, 53, 56
collection objects 23, 29, 40, 50
Commit 112
CONNECT/C++ 94
    .Call interface 95, 97, 99, 100, 102, 107, 113
    Assign 96
    catch block 102
    classes 94, 95, 99, 101, 104, 105, 108, 109, 111, 113, 114
        connection 102
        data object 101
        evaluator 102, 103
        function evaluation 101
    class library 94, 101, 104

Commit 112
constructors 101, 104, 105, 112, 113
converting objects 107
Create 96, 101, 111, 112, 113
CSParray 108
CSPengineConnect 95, 102, 112, 113, 118
CSPevaluator 102, 103, 113
CSPnumeric 95, 107, 108, 111, 112, 114
CSPobject 101, 104, 105, 106, 107, 109, 111, 112, 113, 114
CSPproxy 109
CSPvector 108
destructors 104
generating functions 104
IsValid 104, 111
methods 99, 101, 102, 104, 112
objects
named 111, 112, 113
unnamed 113
OnModify 111, 112, 113
OnRemove 111, 112, 113
operators
arithmetic 109
assignment 104, 105, 106
conversion 107
overloading 106, 108, 109
subscripting 108, 109
printing 111
reference counts 104, 105, 106, 107
Remove 111, 112
S_EVALUATOR 99
s_object 99, 101, 107, 111, 112, 113
sconnect.h 95, 99, 116
SyncParseEval 96
try block 99, 102
try-catch block 99
CONNECT/Java 20
.JavaField function 22, 25, 44
.JavaMethod function 25, 26, 29, 44
class files, Java 26, 27, 42
classpath, Java 26, 27, 28
collection objects 23, 29, 40, 50
descriptors
class 22
field 22, 23, 24, 25
method 25, 26
evalDataQuery function 37
examples
CorrelationExample 41, 44
DataFrameArrayListExample 41, 50
GraphButtonsExample 41, 46
ImageCalculatorExample 41, 50
RandomNormalExample 41, 43
ResultSetExample 41, 48
TextOutputExample 41, 42
extensions, Java 26, 27
FileUtilities 50
getFileInputStream function 50
getFileOutputStream function 50
instance methods 22, 27
instanceof 40
integer conversions 24
Java
Java 2
JDK 20
runtime environment 20
Java Native Interface (JNI) 22
virtual machine 20, 26
java 27, 36
java.graph 36, 41
java.lang.Math 24
java.util.Random 27
javac 28
javaGetResultSet function 48
javap 24, 26
JComponent 41
JFileChooser 28

LocalResultSet 48
nonstatic methods 22
primitives, passing 25
ResultSet 48
ResultSetUtilities 48
scalars, passing 25
*See* Java
special values
Java 24
S-PLUS 24
SplusBadDataException 40
SplusDataResult 37, 38, 39, 40, 41, 44, 50
SplusIncompleteExpressionException 37
SplusSession 37, 50
SplusUserApp 36, 37, 41, 44
static fields 22, 24, 44
static methods 22, 25, 27, 29, 36, 44
StaticMethodsExample 26
System.exit 37
threads, connection 37
tracking instances 22, 27, 29
transferBytes function 50
types, Java 38
winjava library 20, 24, 26, 42
www.javasoft.com 20
connection classes 102
connection threads 37
constructors 101, 104, 105, 112, 113
context-sensitive help 8
conventions, typographic 17
conversion operators 107
conversions, integer 24
converting objects 107
CorrelationExample 41, 44
Create 96, 101, 111, 112, 113
CSParray 108
CSPengineConnect 95, 102, 112, 113, 118
CSPevaluator 102, 103, 113
CSPnumeric 95, 107, 108, 111, 112, 114
CSPobject 101, 104, 105, 106, 107, 109, 111, 112, 113, 114
CSPproxy 109
CSPvector 108
Cygwin 66, 67, 68, 72


## D


DataFrameArrayListExample 41, 50
data object classes 101
decimal.point argument 59
decimal markers 7, 53, 55, 59, 60, 62
Decimal Point field 61, 62
delimiter, semicolon 61
descriptors
class 22
field 22, 23, 24, 25
method 25, 26
destructors 104
digit-grouping symbols 59, 60
displaying numerics 6, 7, 52, 53, 59, 62


## E


evalDataQuery function 37
evaluator classes 102, 103
exportData function 59, 60
exporting numerics 7, 52, 59
Export To File dialog 60, 61
extensions, Java 26, 27


## F


field descriptors 22, 23, 24, 25
file formats, ASCII 61
FileUtilities 50
Find dialog 11
function evaluation classes 101


## G


Gauss-Seidel method 97, 99
General Settings dialog 54
generating functions 104

getFileInputStream function 50
getFileOutputStream function 50
globalization 6
GraphButtonsExample 41, 46
guiImportData function 62

## H

help 8
  ? function 11
  at the command line 11
  context-sensitive 8
  Find dialog 11
  from the graphical user interface 8
  Help button 8
  help function 8, 11
  Help menu 8
  help window
    navigation pane 8, 10
      Contents page of 10
      Index page of 10
      Search page of 10
    toolbar 8, 9
      buttons on 9
    topic pane 8, 11
  in the Script window 11
  keywords 10
  Language Reference 8, 86
  searching in 11
help.log 72, 73
help files, user-defined 7, 64, 65
  .chm file 64, 66, 67, 68, 70, 71, 72, 73
  build_chm_tools 66, 67, 68, 71, 72
  build_helpchm.cyg 67, 68, 70, 72, 73
  Cygwin 66, 67, 68, 72
  help.log 72, 73
  hhgen 72
  HTML Help Workshop 66, 67, 68, 71, 72
  keywords 65, 77, 86, 87, 91, 92
  prompt function 65, 68, 69, 90
  text formats for 74
  titled sections in 76, 90, 91
hhgen 72
HTML Help
  *See* help
HTML Help Workshop 66, 67, 68, 71, 72

## I

ImageCalculatorExample 41, 50
importData function 59, 60
Import From File dialog 60
importing numerics 7, 52, 59
installation 3
instance methods 22, 27
instanceof 40
integer conversions 24
ISO 8859-1 character set 6, 57, 58
IsValid 104, 111

## J

Java
  .JavaMethod function 6
  java.graph 6
  Java 2
    JDK 20
    runtime environment 20
  Java Native Interface (JNI) 22
  virtual machine 20, 26
  *See* CONNECT/Java
java 27, 36
java.graph 36, 41
java.lang.Math 24
java.util.Random 27
javac 28
javaGetResultSet function 48
javap 24, 26
JComponent 41
JDBC 48, 50
JDK, Java 2 20
JFileChooser 28
JNI 22, 23, 24, 25, 26
JRE 20

## K

keywords 10, 65, 77, 86, 87, 91, 92

## L

Language Reference 8, 86
Latin1 52, 57, 58
locales 6, 7, 52, 53, 57, 58
    C 53, 55, 56
    changing 55
    setting 53
        in the Commands window 54
        in the General Settings dialog 55
    Sys.getlocale function 55
    Sys.setlocale function 53, 54, 55
    Sys.withlocale function 55
LocalResultSet 48

## M

manuals, online 11
    viewing 11
markers, decimal 7, 53, 55, 59, 60, 62
method descriptors 25, 26
methods, CONNECT/C++ 99, 101, 102, 104, 112

## N

named objects 111, 112, 113
names, variables
    characters allowed in 57
    characters not allowed in 58
navigation pane, help window 8, 10
    Contents page of 10
    Index page of 10
    Search page of 10
nonstatic methods 22
numerics
    displaying 6, 7, 52, 53, 59, 62
    exporting 7, 52, 59
    importing 7, 52, 59

## O

objects
    named 111, 112, 113
    unnamed 113
OnModify 111, 112, 113
OnRemove 111, 112, 113
operators
    arithmetic 109
    assignment 104, 105, 106
    conversion 107
    overloading 106, 108, 109
    subscripting 108, 109
order, sort 53
order function 56
overloading operators 106, 108, 109

## P

platforms, supported 3
postscript function 58
prompt function 65, 68, 69, 90

## R

RandomNormalExample 41, 43
reference counts 104, 105, 106, 107
Regional Options, Windows 6, 53, 54, 55
Regional Settings, Windows 6, 53, 54, 55
Remove 111, 112
resources 8
ResultSet 48
ResultSetExample 41, 48
ResultSetUtilities 48

## S

S.init file 54, 55
S_EVALUATOR 99
s_object 99, 101, 107, 111, 112, 113
sconnect.h 95, 99, 116
Script window, getting help in 11
search 11
semicolon delimiter 61

sequences, collation 6, 53, 56
Set Region-specific defaults check box 54, 55
Setup, S-PLUS 3, 4, 52
sort function 56
sorting 52
sort order 53
special values
  Java 24
  S-PLUS 24
S-PLUS
  Analytic Server 20, 24, 26
  books using
    graphical techniques 16
    introductory 14
    S-PLUS programming 16
    statistics and data analysis
      general 14
      specialized 15
  installing 3
  running 4
  Setup 3, 4, 52
  special values 24
  starting
    as console application 5
    with graphical user interface 5
  training courses in 13
  Web site 12
SplusBadDataException 40
SplusDataResult 37, 38, 39, 40, 41, 44, 50
SplusIncompleteExpressionException 37
SplusSession 37, 50
SplusUserApp 36, 37, 41, 44
Standard Generalized Markup Language (SGML) 64
static fields 22, 24, 44
static methods 22, 25, 27, 29, 36, 44
StaticMethodsExample 26
subscripting operators 108, 109
support, technical 13
symbols, digit-grouping 59, 60
SyncParseEval 96
Sys.getlocale function 55
Sys.setlocale function 53, 54, 55
Sys.withlocale function 55
System.exit 37
system requirements 3

## T

technical support 13
TextOutputExample 41, 42
thousands.separator argument 59
thousands separators 7
  *See* digit-grouping symbols
threads, connection 37
Tip of the Day 12
toolbar, help window 8, 9
  buttons on 9
topic pane, help window 8, 11
training courses, S-PLUS 13
transferBytes function 50
try block 99, 102
try-catch block 99
types, Java 38
typographic conventions 17

## U

unnamed objects 113
user-defined help files
  *See* help files, user-defined

## V

values, special
  Java 24
  S-PLUS 24
variable names
  characters allowed in 57
  characters not allowed in 58
virtual machine, Java 20, 26

## W

Windows Regional Options 6, 53, 54, 55

Windows Regional Settings 6, 53, 54, 55
winjava library 6, 20, 24, 26, 42
www.javasoft.com 20